U.S. ARMY RDECOM

TECHNICAL REPORT RDMR-BA-14-01

USE OF UNIFIED MODELING LANGUAGE (UML) IN MODEL-BASED DEVELOPMENT (MBD) FOR SAFETY-CRITICAL APPLICATIONS

Susan M. Bonne
Software Engineering Directorate
Aviation and Missile Research, Development, and Engineering Center

And

Jason K. Rupert
APT Research, Inc.
4950 Research Drive
Huntsville, AL 35805

December 2014

AMRDEC

DESTRUCTION NOTICE

FOR CLASSIFIED DOCUMENTS, FOLLOW THE PROCEDURES IN DoD 5200.22-M, INDUSTRIAL SECURITY MANUAL, SECTION II-19 OR DoD 5200.1-R, INFORMATION SECURITY PROGRAM REGULATION, CHAPTER IX. FOR UNCLASSIFIED, LIMITED DOCUMENTS, DESTROY BY ANY METHOD THAT WILL PREVENT DISCLOSURE OF CONTENTS OR RECONSTRUCTION OF THE DOCUMENT.

DISCLAIMER

THE FINDINGS IN THIS REPORT ARE NOT TO BE CONSTRUED AS AN OFFICIAL DEPARTMENT OF THE ARMY POSITION UNLESS SO DESIGNATED BY OTHER AUTHORIZED DOCUMENTS.

TRADE NAMES

USE OF TRADE NAMES OR MANUFACTURERS IN THIS REPORT DOES NOT CONSTITUTE AN OFFICIAL ENDORSEMENT OR APPROVAL OF THE USE OF SUCH COMMERCIAL HARDWARE OR SOFTWARE.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 074-0188

1. AGENCY USE ONLY	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	December 2014	Final

4. TITLE AND SUBTITLE
Use of Unified Modeling Language (UML) in Model-Based Development (MBD) for Safety-Critical Applications

5. FUNDING NUMBERS

6. AUTHOR(S)

Susan M. Bonne and Jason K. Rupert

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Commander, U.S. Army Research, Development, and Engineering Command
ATTN: RDMR-BAV
Redstone Arsenal, AL 35898-5000

8. PERFORMING ORGANIZATION REPORT NUMBER

TR-RDMR-BA-14-01

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION / AVAILABILITY STATEMENT

Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE

A

13. ABSTRACT (Maximum 200 Words)

One of the key assumptions behind most Model-Based Development (MBD) efforts is the selection of Unified Modeling Language (UML) for the design language. While popular, more than 90 percent of all MBD efforts choose UML. This choice is often taken for granted. Typically, no trade studies are provided to support the choice of UML, especially the evaluation of UML traits that would make it adequate for use in MBD for safety-critical applications. Given that UML is selected for MBD, this report seeks to look at some of UML's traits in light of safety related expectations. Moreover, this report recommends practices for using UML on safety-critical applications and makes an appeal to the MBD community for additional suggested safe practices.

14. SUBJECT TERMS
Software, Unified Modeling Language (UML), Model, Safety Critical, Flight Critical

15. NUMBER OF PAGES
78

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	SAR

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. Z39-18
298-102

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

I. INTRODUCTION

Unified Modeling Language (UML) appears to be ubiquitous with software design for efforts requiring minimal maturity to programs with mandatory regulatory rigor [1]. On programs using Agile processes [2, 3] where minimal formal documentation is mandated, UML is highlighted as a good communication tool to use between all stakeholders. UML also continues to gain a foothold in larger scale programs where spiral, waterfall, mature Agile [4], or other mature practices are used, for example, Rational Unified Process (RUP) [5]. Beyond that, UML is used in safety-critical applications from the automotive to medical industries and flight critical fields from manned to unmanned applications. Moreover, college and university courses use UML to help teach software and system design principles, as shown in Reference 6. Thus, it is easy to see that there is something easily accessible and appealing about representing design elements using graphical symbols.

With the growth of Model-Based Development (MBD), especially with the ability to generate code from the design, the software life cycle will make great use of UML. UML will be used to represent system models, subsystem models, and even some low-level models. When thinking about a waterfall software life cycle, both the early and middle steps in the software life cycles could be represented by models, for example, systems and high-level requirements development phases. Although not fully realized today, the direction seems to be to produce fewer stacks of paper documents and have the model be the artifact or be able to be used to produce the artifacts. Naturally, this produces fewer stand-alone paper artifacts and makes greater use of a centralized model. For example, with MBD, a stand-alone system requirements document will exist as a System Modeling Language (SysML) model. SysML is a UML profile. A SysML model could then be used to automatically produce the appropriate system level artifacts, for example, System Subsystem Design Document (SSDD).

The use of MBD and UML must meet the needs of the safety and flight certification authorities. MBD must work with certification authorities to show equivalency between MBD artifacts and those artifacts expected by certification authorities. Additionally, MBD must be able to demonstrate rigor and maturity by producing artifacts required for the appropriate certifying authority. For example, most safety authorities will require some demonstration of bi-direction traceability, showing that standards were followed and verification and validation was accomplished. This report lists some concerns with using UML in safety-critical environments, and some concerns about UML's ability to meet the needs of certification authorities. However, the report also provides some suggested practices to address those concerns. Recognizing that this report is not a comprehensive list of good practices, this report makes an appeal to the MBD community for additional suggested practices and considerations for the use of UML in safety-critical applications. While safety-critical terminology is used throughout this report, the concepts presented are also applicable to flight critical applications. In order to help facilitate access to the material presented in this report, a briefing package was assembled and is provided in the appendix.

II. BACKGROUND

A. Model

What are models? The definition of models is as varied as the subjects being represented by them. The use of models varies from the highly abstract computational intensive to intricate representations of complex physical phenomena. Likewise, there are definitions of the term model to justify each of those uses being called a model. For the purpose of this report, two definitions of model are presented. The first is provided by Radio Technical Commission for Aeronautics (RTCA) and found in Reference 7. Reference 8 provides the following definition of model:

"An abstract representation of a given set of aspects of a system that is used for analysis, verification, simulation, code generation or any combination thereof. A model should be unambiguous, regardless of its level of abstraction" [8].

Moreover, to be unambiguous, the model should provide sufficient detail to describe and verify the desired response to both normal and abnormal stimuli.

The RTCA definition was provided because this report targets the use of UML models in safety-critical environments. RTCA provides forums that help produce guidance materials for avionics, specifically performance standards and considerations for avionics equipment certification. Most notable for this report is that RTCA is shepherding the generation and update of RTCA Document (DO) 178.

Likewise, for the context of this report, a definition of model provided by the Object Management Group® (OMG®) is equally valuable. Unfortunately, the definition of model provided by OMG® seems to vary from source to source [9, 10, 11], so the following one from Reference 9 has been selected:

"Models in the context of the [Model Driven Architecture] MDA Foundation Model are instances of [Meta-Object Facility] MOF™ metamodels and therefore consist of model elements and links between them. This required MOF compliance enables the automated transformations on which MDA is built. UML compliance, although common, is not a requirement for MDA models. (This means, for example, that a suitable development process based on OMG's Common Warehouse Metamodel [CWM] can be MDA-compliant, since CWM is based on MOF.)" [9].

It is unfair to judge the RTCA definition against the OMG® definition, but they should be compared. RTCA provides a descriptive definition, while OMG® provides a prescriptive definition. For OMG®, in order to be considered a model, it must be founded on MOF™. Note that the MOF™ specification is managed by OMG® [12] and is also an international standard [13]. A full explanation of MOF™ is beyond the scope of this report (the specification is 88 pages), but it can be summarized this way: MOF™ is the architecture underlying MBD and thus UML.

B. Model-Based Development

For this report, an applicable definition of MBD is provided in RTCA DO-331 [8]:

"A technology in which models represent software requirements and/or software design descriptions to support the development and verification process."

Note that this definition acknowledges that requirements and designs are represented within the model, but it is curious that this definition is limited to "software requirements and/or software design" [8] being represented in the model. It is speculated that the limitation to only software is due to the hierarchy of avionic specifications, that is, SAE ARP4754A [14] and SAE ARP4761 [15] address system level avionics concerns.

In order to push the discussion of MBD further, a discussion of the architecture used within MBD is necessary. Again, refer to OMG® [1]:

"The MDA is a new way of developing applications and writing specifications, based on a platform-independent model (PIM) of the application or specification's business functionality and behavior. A complete MDA specification consists of a definitive platform-independent base model, plus one or more platform-specific models (PSM) and sets of interface definitions, each describing how the base model is implemented on a different middleware platform. A complete MDA application consists of a definitive PIM, plus one or more PSMs and complete implementations, one on each platform that the application developer decides to support" [1].

Moreover,

"Any modeling language used in MDA must be described in terms of the MOF language, to enable the metadata to be understood in a standard manner, which is a precondition for any ability to perform automated transformations" [1].

Unfortunately, the concept of the PIM and PSM are a bit different than the concept of specification and design model discussed in RTCA DO-331. A specification model represents a high-level view of the software component. This can be independent of the Hardware (HW), like a PIM, or it can include details of the target HW. For RTCA DO-331, the design model provides the low-level requirements for a software component, while a PSM may include the details for a software component or the whole system. These differences pose some challenges when comparing the expectations within RTCA DO-331 with the Model Driven Architecture (MDA®) approaches laid out by OMG®. The formal MBD/MDA® definitions provided by RTCA and OMG® are important, but it can also be described as the following:

"Model-driven development is simply the notion that we can construct a model of a system that we can then transform into the real thing" [16].

C. Unified Modeling Language

UML is a General Purpose Modeling Language (GPML) that uses a set of graphic notations to specify, constrain, and document visual models. Like the MDA®, UML is managed

by OMG® and is an International Organization for Standardization (ISO) standard. OMG® UML is described in References 17 through 20.

The Object Constraint Language (OCL) specification is most applicable to the topics covered in this report. However, both the infrastructure and superstructure documents indicate the following UML trait:

"Not all of its modeling capabilities are necessarily useful in all domains or applications. This suggests that the language should be structured modularly, with the ability to select only those parts of the language that are of direct interest" [17].

OCL brings a formal constraint language to UML expressions. The constraints added by OCL are invariant conditions and do not have side effects, that is, they cannot alter the executing state or values of the model. For example, for the Ground Control Station (GCS) to specify a precondition constraint that the number of Unmanned Air Vehicles (UAVs) under control must be greater than or equal to 1 has the following syntax: `self.NumControlUA >= 1`. Mature UML tools treat the OCL as a query language and thus make queries on the UML model. Being a formal language by design, OCL lacks some of the ambiguities associated with natural languages, so it can be used to clearly describe required safety guard conditions and exit threshold inhibits.

Most people do not immediately think about OCL when considering UML. Most people think diagrams, which, as shown in Figure 1, are divided into two major categories: structural (also known as static) or behavioral (also known as dynamic).

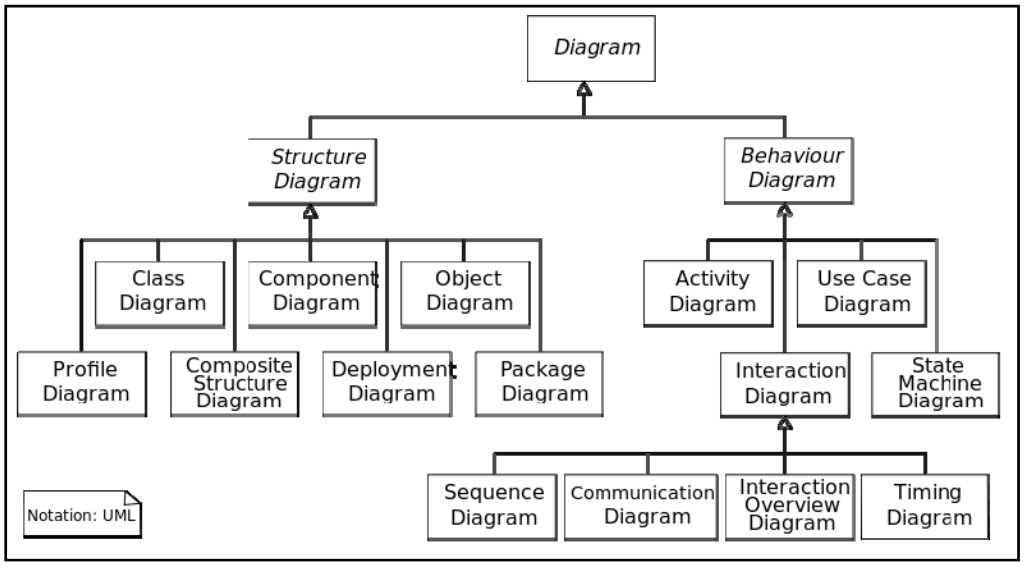


Figure 1. UML Diagram Overview [21]

For the purpose of this report, only one example from each category is discussed. From the static diagram category, the class diagram is discussed, and from the dynamic category, the state machine diagram is discussed. Despite only representing static features, as shown in Figure 2, a class diagram can show the following features: attributes (with type, visibility, and default), operations (with input and return type), inheritance, and associations. Extensive

explanation of all of the features of the class diagram are beyond the scope of this report; however, there are extensive introductory and expert discussions available [22, 23].

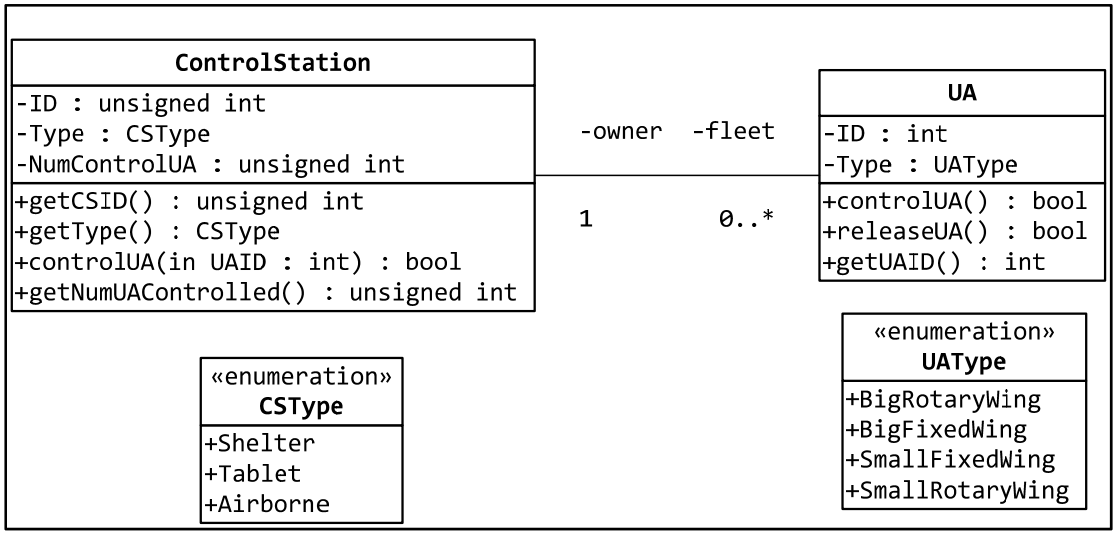


Figure 2. Class Diagram

In Figure 2, the `ControlStation` class has three attributes (`ID`, `Type`, and `NumControlUA`) and four operations (`getCSID`, `getType`, `controlUA`, and `getNumUAControlled`). Likewise, notice that each attribute has an indicated type, for example, `Type` is a `CSType`, which is an enumeration. UML does not require that all types be listed for attributes, that is, unspecified types are allowed.

Figure 3 presents a simple example of a state machine diagram of a `ControlStation` switching between control and monitor states. The black dot indicates the initial transition. Once in the monitor state, the `ControlStation` can remain in that state or transition to the control state. Then, from the control state, the `ControlStation` can remain there or transition back to the monitor state.

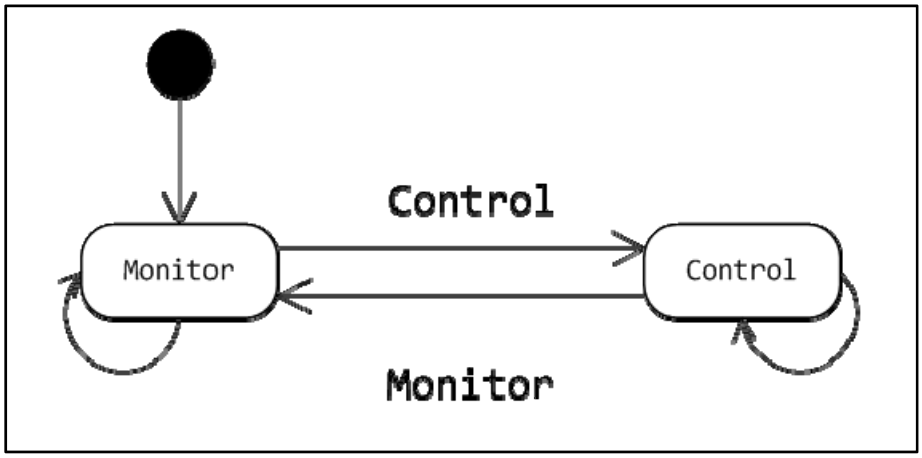


Figure 3. State Machine Diagram

Constraints can be added to clarify the class and state machine diagrams. Consider the additional natural language requirement placed on the class diagram shown in Figure 2: A `ControlStation` (owner) shall control three or fewer UAV. To capture that constraint in UML, the following OCL code would be added to the model:

```
context:    ControlStation
inv:        self.NumControlUA<=3 or self.fleet->size<=3
```

Likewise, OCL could be used to specify that all UAVs controlled by the `ControlStation` are `BigFixedWing` UAVs:

```
context:    ControlStation
inv:        self.fleet->forAll(av.Type=#BigFixedWing)
```

In the previous OCL code, `context` indicates the elements (for example, the class), `inv` indicates the invariant statement, and `self` indicates the current context.

Likewise, as shown in Figure 4, constraints can be added to a state machine. For even a simple example, the constraints necessary to capture the appropriate level precision become involved. For example, the `ControlStation` can only transition to the control state if the `NumControlUA` is greater than zero. Moreover, once in the control state, the `ControlStation` can reduce the number of UAVs controlled by calling `requestReleaseUA` or control an additional UAV by calling `requestControlUA`. Then, the `NumControlUA` value is either reduced or increased, respectively. For either of those cases, the `NumControlUA`, as shown in the square brackets, is checked against the appropriate guard condition, that is, greater than zero or less than or equal to three, respectively.

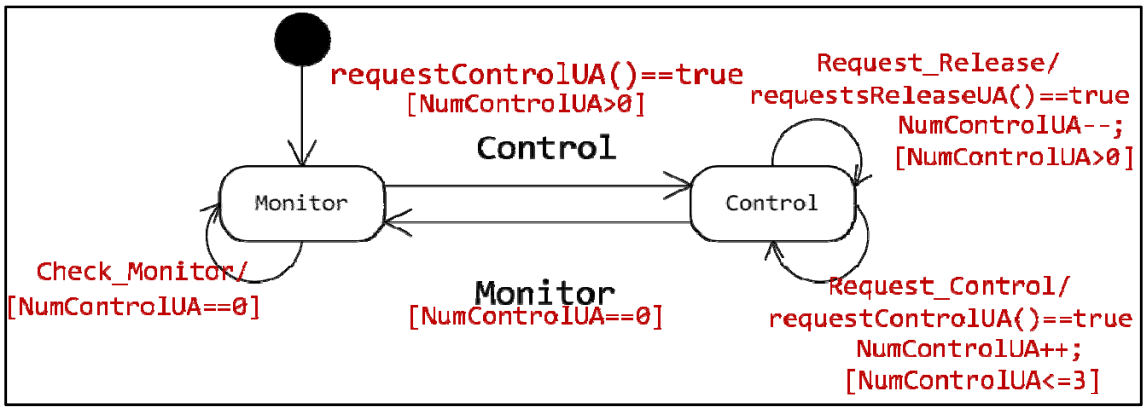


Figure 4. Constraints Added to State Machine Diagram

Extensibility, as described in the OMG® UML superstructure, is one of the key characteristics of UML Version 2. As with previous versions of UML, UML Version 2 still allows the use of tagged values, constraints, and stereotypes as a standard extension mechanism (also known as standard stereotypes). Tagged values capture additional information from administrative (for example, process details) or to architectural (for example, details to help with PIM to PSM translations) and can be attached to various UML graphical elements. While most UML extensions will take advantage of the standard extensions, other core UML extensions can

be made by modifying the underlying MOF™. One way to take advantage of the standard extension is to develop a UML profile.

UML profiles are analogous to spoken language dialects. For example, Cajun is a dialect of North American English. UML profiles can be composed of a namespace and limiting or expanded list of stereotype names, modeling conventions, constraints, tagged values, and other UML conventions. Profiles are created to tailor the GPML features of UML so that the language is targeted at a specific domain of users. For example, SysML is targeted at systems engineers.

While revisions of UML continue to enhance the precision of the syntax, semantics, and definitions, there still exist some cases where language constraints, extensions, and best practices are necessary to qualify UML to be used in safety-critical environments.

D. Safety and Safety-Critical Software

The field of safety is wide, and there are numerous definitions of safety. Reference 24 uses the following definition:

"Safety is defined as the freedom from those conditions [hazards] that can cause death, injury, illness, damage to or loss of equipment or property, or environmental harm" [25].

According to MIL-STD-882E, a hazard is defined as follows:

"A real or potential condition that could lead to an unplanned event or series of events (i.e. mishap) resulting in death, injury, occupational illness, damage to or loss of equipment or property, or damage to the environment" [26].

Next, MIL-STD-882E defines a safety-critical item, either software or HW, as the following:

"Hardware or software that has been determined through analysis to potentially contribute to a hazard with catastrophic or critical mishap potential" [26].

Further, National Aeronautics and Space Administration (NASA) Standard (STD) 8719.13B states that safety-critical software resides in a safety-critical system and has at least one of the following characteristics:

- "Provides controls or mitigations for a hazard
- Controls safety critical functions
- Processes safety critical functions
- Detects and reports, or takes corrective action, if the system reaches a specific hazardous state
- Mitigates damage if a hazard occurs
- Resides on the same system (processor) as safety critical software AND is NOT partitioned from non-essential software" [27].

Once software is identified as being safety critical, depending on the industry, various regulations and guidance material must be followed. In general, that material requires an elevated rigor in the software life cycle process. One example of elevated rigor is that a requirement for software life cycle standards be established and followed and proof (artifacts) be provided that the standards were followed. For example, RTCA DO-178C [7], NASA-STD-8719.13B [27], and International Electrotechnical Commission (IEC) 61508 (specifically, Parts 3 [28] and 7 [29]) all require coding standards. Because government and industry software safety guidance requires the existence of coding standards, various coding standards have emerged: Joint Strike Fighter (JSF), C++ coding standard [30]; Motor Industry Software Reliability Association (MISRA), C [31] and C++ [32] standards; and Jet Propulsion Laboratory (JPL), Institutional Coding Standard for the C Programming Language [33].

Unfortunately, a coding standard alone in a safety-critical software life cycle has several weaknesses, especially in the context of MBD:

- Verification against coding standards occurs after software development, so the application of the coding standard and verification of the accurate application of the standard come too late in the software life cycle. As noted by multiple studies and reports [34, 35], the later in the software life cycle issues are identified, the more costly they are to fix. For example, some studies have shown costs rise exponentially, as shown in Figure 5, with time [36, 37].

- Typical coding standards only address concerns that are discovered by static analysis of the code. That is, typical coding standards capture criteria that could be checked during manual peer reviews (if the reviews are rigorous). For example, coding standards do not address logical errors or other coding weaknesses that are caught during dynamic analysis or rigorous testing.

- Typically, the coding standard does not specifically address error handling, fault detection, traceability, or verification.

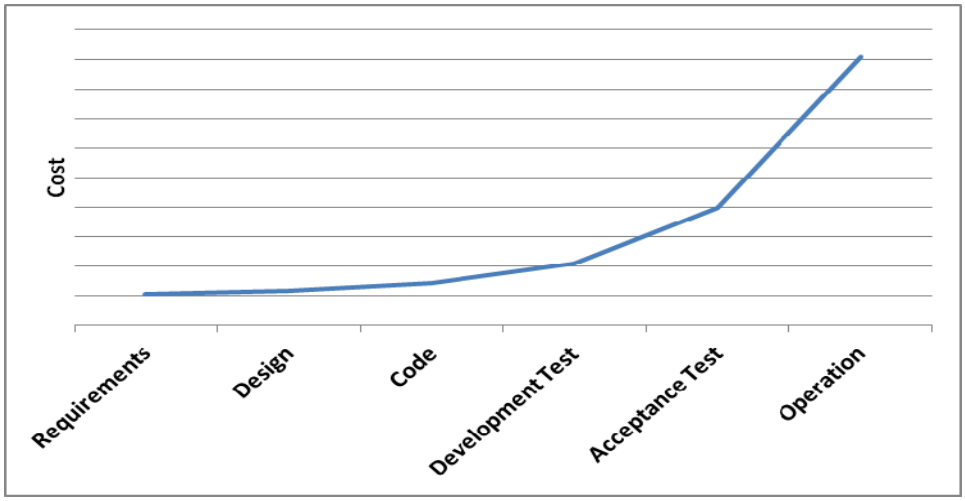


Figure 5. Boehm Curve [36]: Relative Cost to Fix Error Versus Phase in Which Error Detected

Given those weaknesses, requirements and design standards, coverage analysis, and other verification procedures (for example, data and control coupling) are also required for safety-critical applications. Of these, this report addresses the importance of the requirements and design standard, where the design standard, as will be discussed in the following sections, is especially important for safety-critical MBD. With the use of the design standard in MBD, those weaknesses previously listed can be addressed earlier in the life cycle.

III. APPLICATION

A. Unified Modeling Language in Model-Based Development

The implementation of MBD can take various approaches, for example, structural, behavioral, abstraction, automation, and so forth. In addition to the various approaches, MBD also does not specify the use of a Domain Specific Modeling Language (DSML) like Simulink® or a GPML like UML. Of course, in order to be compliant with the OMG® MDA® standard, the chosen MBD language must be conformant with MOF™. Regardless or because of the OMG® desire for all MDA®/MBD languages to be conformant with MOF™, it appears that UML is becoming the de facto MBD language. OMG® states the following on the MDA® FAQ page:

"Although not formally required [for MBD], UML is still a key enabling technology for the Model Driven Architecture and the basis for 99% of MDA development projects" [1].

Without finding anything to the contrary, it is assumed that the previous statement is also true for safety-critical MBD/MDA® efforts. The reason for the limited use of DSMLs, like SAE Architecture Analysis and Design Language (AADL), on safety-critical MBD efforts remains unclear. However, MBD tools like Rhapsody, MagicDraw, or Atego Artisan Studio do not support AADL and other DSML, which is likely due to lack of market forces. There appear to be efforts to port AADL to a UML profile, but discussion of the status of those efforts is beyond the scope of this report [38, 39].

B. Models in Model-Based Development

Given the innate characteristics of UML to be extended, subsetted, and for profiles to be developed, the application of UML seems to be limited only by the creativity of the software designer or system architect. However, one limitation of early versions of UML was the ability to represent requirements. In order to address this limitation and exercise the characteristics of UML, the International Council on Systems Engineering (INCOSE) established the SysML profile, which is a UML profile developed by subsetting and extending UML. One of the main goals of SysML was to establish requirements as "first-class model elements" [40]. Doing this allowed for the generation of a requirements table, that is, "traceability information for requirements in a single view" [40].

The SysML profile of UML targets the systems engineering discipline, specifically Model-Based Systems Engineering (MBSE). Being a systems engineering profile, it tailors/extends some of the software engineering characteristics of UML. For example, UML uses *DataType*, while SysML uses *ValueType*, which is more neutral. Like UML, SysML has structure and behavior diagrams. In addition to those types of diagrams, SysML adds the requirement diagram, which is used to "display text-based requirements, the relationships

between requirements and the other model elements that satisfy, verify, and refine them" [41]. The requirement diagram merges the traditional textual requirement with graphical features of MBSE, for example, showing traceability between model elements and textual requirements. Those traces can then be used to "autogenerate requirement traceability and verification matrices (RTVMs)" [41]. Relationships in requirement diagram are typically shown using one of the following: containment, trace, derivative, refinement, satisfy, or verify. The satisfy relationship, that is, «`satisfy`», shows the linkage from a requirement element to a block element. This linkage is for illustrative purposes, so it must still be verified.

The MBD/MDA® approach discussed by OMG® involves translations from PIMs to PSMs, that is, the PIM to PSM translation. The basic meaning behind each of these models is as follows:

- PIMs—A representation of the system or subsystem or component that is abstracted from the deployment target.
- PSMs—A representation of the system or subsystem or component that includes adequate detail for target deployment.

The translation from PIM to PSM involves refinement of the abstract PIM. This refinement is complete when enough details have been added to the PSM that it can be deployed on the intended target.

Sometimes during this PIM to PSM translation and during the PSM refinement process, using UML to represent low-level and algorithmic behavior (business logic) is difficult. At this point, architects or designers insert code or DSML into UML, for example, Matrix Laboratory (MATLAB®)/Simulink® or SCADE. This is a known limitation [42], so UML users should plan accordingly, especially for safety-critical programs. For example, "models are not used to compute functions, be it numeric (such as factorials) or symbolic (such as text processing)" [43]. Other examples from other major industry players are the following:

- "…the developer actually write[s] the code contained in the state machines. This code provides the "algorithmic details" that actually make the application work" [44].
- "Behavior code is manually input into the Platform Specific Model (PSM) using C++" [45].

For safety-critical programs, insertion of behavior code into the model will have to be monitored very carefully as this is a blending of the design and coding processes. Blending the two separate software processes must be shown to meet all the appropriate criteria, for example, transitional criteria between life cycles, traceability of safety-critical requirements, abiding by both design and coding standards, and verification of the behavior code, design, and low-level requirements.

IV. UNIFIED MODELING LANGUAGE IN MODEL-BASED DEVELOPMENT FOR SAFETY-CRITICAL APPLICATIONS

Given the extent to which UML is being used and will continue to be used for MBD and the likelihood of UML being used on safety-critical MBD efforts, the goal of this report is to provide a short list of initial concerns with UML, start the discussion regarding the development of suggested practices, and make an appeal to the MBD community for help developing UML guiding practices for safety-critical applications.

A. Concerns

The following is a list of concerns regarding the use of UML on safety-critical MBD efforts:

- Informal
    - "UML is not a methodology, it does not require any formal work products" [46].
    - "A UML diagram, such as a class diagram, is typically not refined enough to provide all the relevant aspects of a specification" [18].
    - "All models, both PSM and PIM, should be consistent and precise, and contain as much information as possible about the system. This is where OCL can be helpful, because UML diagrams alone do not typically provide enough information" [47].
    - UML does not enforce rigor or completeness.
- Consistency
    - "UML cannot fully define the relationships between diagrams. The diagrams are developed as separate entities that express different aspects of the software, not as parts of a common construct" [48].
    - "…the current UML spec really does not restrict graphical formats in any way -- it simply provides a standard set of notations, but not at the exclusion of other notations. In other words, there really is no "illegal" UML graphical syntax" [49].
- Extensibility—As previously mentioned, by design, UML is extensible. As long as the new features are conformant to the standards, they can be added ad nauseum: stereotypes, elements, graphics, tags, and so forth.
- Questionable Behavior—"In UML, active objects have their own thread of control, and can be regarded as concurrent threads [with possible unintended safety consequences]. Only extensions of the UML standard, such as the MARTE [Modeling and Analysis of Real Time and Embedded systems] profile, provide mechanisms to model detailed information pertaining to concurrency" [50].

Previously mentioned was the lack of a formal way to represent requirements, tag those requirements as safety critical, and then trace those requirements into high-level and low-level design. Also mentioned was the weakness with only applying a coding standard during the software life cycle.

B. Suggested Practices

After listing some concerns with using UML in a safety-critical environment, the following four sections list some practices for addressing some of those concerns:

- Develop and Document UML Design Standards
- Select Appropriate Tool Chain
- Conduct Reviews, Verification, and Validation
- Develop Safety-Critical Profile (Action for the Safety Community)

1. UML Design Standard

The first task is to develop a UML design standard that includes general good practices as well as formal rules. Within the UML design standard, in accordance with RFC2119 [51], consider using three levels of imperatives for both good practices and rules: shall (absolute), should (absolute unless justified in particular circumstance), and may (truly optional). Understandably, the UML design standard may need to be tailored to the individual program, but a core list of UML specific design standards should be captured.

First, the UML design standard should address good practices. There are some examples of these in Reference 52, where the first good practice listed is "Apply a subset of UML relevant to your role." Likewise, DO-331 Section 11.23 provides a list of good practices that should be included in a UML design standard, for example, maximum number of models per diagram. Along these lines, since Object-Oriented (OO) technologies are used in UML MBD, good OO design principles should be applied: inheritance, polymorphism, abstraction, and so forth. Additional OO considerations for safety-critical environments are covered in Reference 53. That document and others provide good practices; however, they are not prescriptive enough to be considered formal rules.

Second, the UML design standard should develop formal binding rules for UML that shall be followed for all safety-critical applications. Those rules should cover all things related to the use of UML to represent safety-critical designs from style (for example, graphical design indicators such as thick red border for all safety critical elements [27]) to classes (for example, not allowing unbounded multiplicity or anonymous instances for classes) and no concurrent states for state diagrams.

The rules should specify the approach that should be taken for a UML model to capture constraints and expressions. According to Reference 54, constraints are "a restriction on one or more values of (part of) an object-oriented model or system." Constraints can come as either informal additions, for example, natural language comment, or as a formal constraint such as those applied by the OCL. Given the formal declarative nature of OCL which is part of the UML Specification, it is preferred over the informal nature of natural language comments. The

UML design standard rules should detail how invariants, preconditions, and post conditions of operations should be specified in the model. Use of these rules should help make the UML diagrams more precise and if properly implemented, reduce some of the UML concerns, as shown in Figure 4.

Third, the UML design standard should address traceability. NASA-STD-8719.13B [27], RTCA DO-178C [7], MIL-STD-882E [26], AMCOM 385-17 [55], and other software safety regulations discuss the importance of traceability, especially the traceability of safety-critical requirements. "Requirements traceability is defined as the ability to describe and follow the life of a requirement in both directions, towards its origin or towards its implementation, passing through all the related specifications" [56]. Note that one of the updates from RTCA DO-178B [57] to RTCA DO-178C [7] was to clarify the need for bi-directional traceability artifacts, that is, trace data. Moreover, RTCA DO-331 [8] states that when MBD/MDA® is used to represent requirements, as with SysML, the modeling standard for the purpose of this report, the UML design standards, should address rules for representing requirements and also bi-directional traceability tags or markers. Given that the intent of SysML is to capture requirements, the UML design standard must address requirements (including derived requirements) related rules that must be followed in the model. For example, all requirements should state what the system does. Avoid, unless justifiable, negative requirements as they are difficult to verify.

- Not allowed—Red shall not be used in standard text. (This does not specify what color shall be used.)
- Allowed—Red (650 nanometers plus or minus 10 nanometers) shall only be used for alarm messages.

For some initial guidance on best practices for requirements standards, refer to Reference 58.

Model requirements may be linked with functional behaviors, so the UML design standard must identify the method used to link the requirements to the model. This specification must allow bi-directional trace data to be produced. It must support requirements represented within or external to the model and textual or graphical requirements. Figure 6 shows an example of how this trace may look when combining SysML with a UML class diagram. Certain modeling tools can use the «`satisfy`» stereotype and produce/output trace data artifacts. Notice that the «`deriveReqt`» stereotype is used to indicate derived requirements. Various modeling tools have ways of addressing external linkages if the requirements are external to the model. For example, Atego's Artisan Studio uses SysML to allow the Dynamic Object-Oriented Requirements System (DOORS®) requirements module to synchronize with the model. Regardless of the location of the requirements or their format, for safety-critical items, the UML design standard should address, as appropriate, the method used to trace requirements from systems to high-level requirements, high-level to low-level requirements, and low-level to code. Likewise, traceability from requirements to test should also be addressed. Certain modeling tools can use the «`satisfy`» stereotype and produce/output trace data artifacts [59].

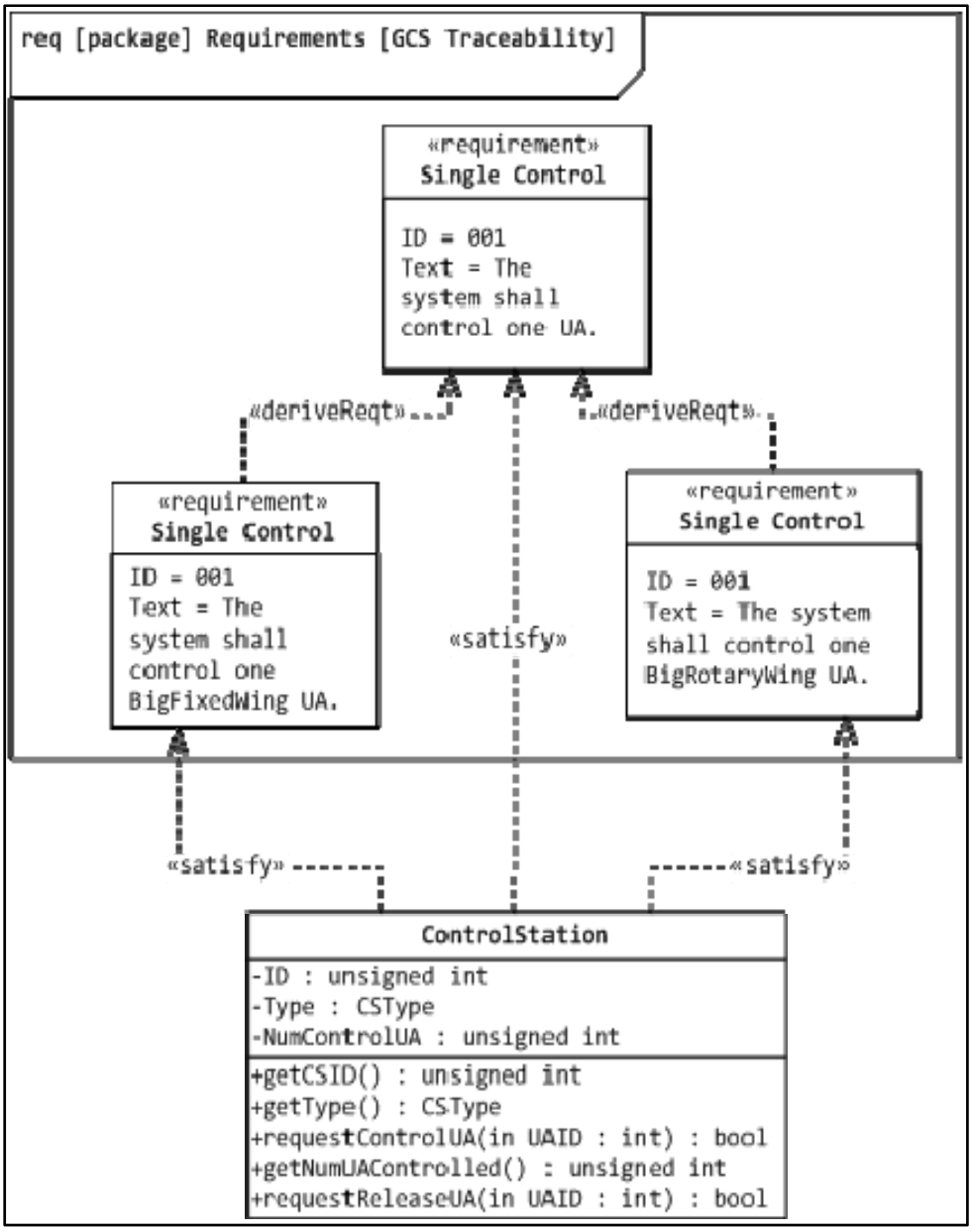


Figure 6. SysML Requirements and Traceability Using <<satisfy>>

Fourth, the UML design standard should address other language semantics, for example, capture the use of any extensions, unique stereotyping, or non-normative elements. By design, UML can be extended, so if any extensions are used, those should be described in the UML design standard. For example, by default, UML 2.1 contains 28 stereotypes. SysML adds five new stereotypes. Include the appropriate rules to state if any of those stereotypes are prohibited or new stereotypes are added, for example, «`safetyCritical`». Moreover, document the approach to including Non-Functional Parameters (NFPs) and Non-Functional Requirements (NFRs) in the model, for example, safety, security, reliability, and performance. Figure 7 shows how an NFP can be attached to a class diagram.

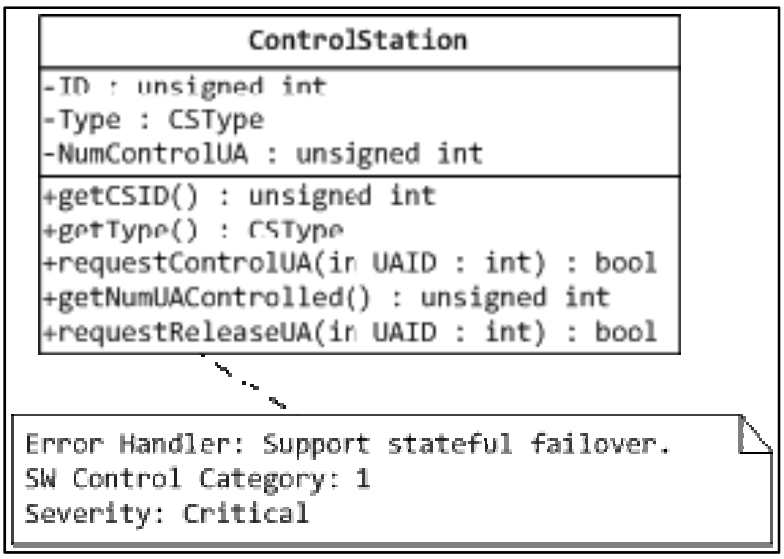


Figure 7. NFP Note Attached to Class Diagram

2. UML MBD Tool Chain Selection

Like the importance of selecting tools for traditional software development for safety-critical applications, the selection of UML modeling tools is equally important. A full listing of UML tools or adequate tool qualities is beyond the scope of this report. It is the goal of this section to reinforce the importance of selecting a mature tool that is appropriate for the criticality of the target applications. For example, Dia [60], a diagramming tool, may be appropriate for prototyping efforts. Because it is no longer being maintained (no updates since 2011) and has limited functionality, Dia may not be appropriate for the development of a safety-critical release candidate.

If multiple MBD/MDA® tools are being used, tool compliance level is very important. That is, choose tools that support the highest model interchange compliance level, as defined in the OMG® UML superstructure, infrastructure, and OCL specifications. Complications may result even if the tool has high compliance, and insurmountable obstacles may result otherwise.

What follows is a list of other UML MBD tool evaluation criteria to consider:

- Support latest and legacy UML versions
- Support full UML specification, for example, extensions and OCL.
- Provide extensibility interface for user-defined plug-ins
- Interface with domain specific models or domain specific languages
- Perform inter-diagram consistency verification, that is, verifies consistency among diagrams
- Perform static/structural constraint verification
- Perform dynamic/behavioral constraint verification
- Qualifiable mature tool, for example, according to DO-330 tool qualification

- Produce certifiable code
- Support profiles (for example, support SysML)

Dated UML tool evaluation results are provided in Reference 61.

In addition to applying judicious tool selection criteria, once the selection(s) are performed, a full description of the MBD tool chain can be produced. That description could be resident in a Plan for Software Aspects of Certification (PSAC) or Software Development Plan (SDP). Hopefully, that description also includes a brief discussion of the rationale for the selection of the tool(s).

3. Reviews, Verification, and Validation

One of the goals of MBD/MDA® is to help reduce the effort required during the software life cycle (by reducing the number of steps and also level of effort) but not reduce the robustness of the final product. One of the ways to continue to maintain the rigor associated with MBD is to conduct model reviews, verification, and validation, especially for MBD for safety-critical applications. Manual model peer reviews, informal and formal, should be conducted early and often to catch high-level logical modeling errors before low-level modeling is begun. For evaluation criteria, model peer reviews should use the good practices design rules and recommendations in the UML design standards. Manual peer reviews are very effective at identifying missing good practice techniques and some logical errors, for example, good use of abstraction, modularity, cohesion, and coupling, which automated static and dynamic model reviews have difficultly identifying.

In addition to manual peer reviews of the model, conducting automated static and dynamic review of the model is also recommended. There are numerous techniques that can be used for both static and dynamic analysis, but checking model consistency is critical. One of the strengths of UML is the numerous diagrams and various attributes, constraints, stereotypes, and extensions. Unfortunately, this makes evaluating consistency amongst these various structural (for example, class diagram) and behavioral (for example, state diagrams) diagrams difficult. Given that the model is targeted for safety-critical applications, overall model consistency is critical and must be determined.

In addition to consistency checking, many other UML static and dynamic analysis techniques exist. Refer to Reference 62 for a list.

Expressions written in the OCL are used to specify integrity constraints of the model so that OCL constraints can be automatically verified during dynamic analysis. Moreover, during dynamic analysis, OCL expressions can be entered and evaluated to query detailed information about a system state.

Discussion of review, verification, and validation would not be complete without mentioning formal model checking, which is important to the development of software-intensive systems with safety risks, for example, automotive [63] and medical industry [64]. ISO 26262 [65] recommends model checking for both Automotive Safety Integrity Level (ASIL) C and D, the top two risk levels for automotive safety. Model checking is a verification method that

automatically and exhaustively checks that a model meets a given specification. According to NASA-GB-8719.13 [66], model checking checks for the following:

- Reachability (Does a system ever reach a certain state?)
- Lack-of-deadlock (Is deadlock avoided in the system?)
- Safety (Nothing bad ever happens.)
- Liveness (Something good eventually happens.)

Due to most UML modeling tools not including formal model checking, to be checked, UML models must be transformed into formal model checking language. However, Simulink® Design Verifier™ includes some built-in model checker-type capabilities, for example, formal methods [63].

4. Safety-Critical UML Profile

UML and current UML profiles seem to be ill-equipped to address safety-critical needs. For example, after some initial evaluation, the MARTE UML profile does not capture requirements or requirements traceability. As previously mentioned, AADL, currently only a domain specific language, could augment the MARTE profile. Efforts are underway to develop an AADL UML profile, but that effort seems to still be immature and is only targeted at the aviation industry [38, 39].

Although UML profiles are designed to be used together, some deconfliction may be necessary when identical key words or other similar features are used. For example, SysML and MARTE both use `FlowPort`, but the semantics are different [40]. Note that SysML Version 1.3 deprecates FlowPort, but it was present in SysML Version 1.2.

Moreover, this reference further elaborates on the complexity of possibly combining the SysML and MARTE profiles. Mitigations to merging UML profiles are addressed in Reference 67, while difficulties in combining SysML and MARTE are addressed in Reference 68.

As previously mentioned, OMG® UML superstructure [19] Section 18.1.2 Profile Design Requirements provide some guidance on the characteristics of UML profiles:

- "Well-formedness," for example, "constraints that are more constraining (but consistent with) those in the reference [UML] metamodel" [19] (that is, only extensions to the metamodel)
- Using UML XMI it must be possible to interchange profiles between tools
- Reference domain-specific UML libraries
- Combine UML profiles and model libraries via extensions
- Specialize semantics of standard UML elements, for example, restrict to single inheritance without having to assign explicit stereotype to each and every instance
- Profiles can be dynamically applied or retracted from a model

A Safety-Critical profile may be formed by extending a current profile (for example, MARTE), combining some existing profiles (for example, MARTE plus SysML) or creating a new profile specifically applicable to the safety-critical domain; for example, pieces of SysML plus pieces of MARTE plus safety and airworthiness extensions and constraints [40].

Figure 8 shows a sketch of the potential design space for a Safety-Critical profile. Security profiles, for example, UMLSec, are not shown in Figure 8 because too little is known about them to know which portions of those profiles could be leveraged. Quality of Service (QoS), which describes signaling speed, maximum turnaround time, data size, and disconnect threshold, should also be included in the Safety-Critical profile (or individual element characteristics of that profile could be added as necessary). Some effort is being undertaken to develop a DO-178 Airworthiness UML profile, but for this report, that work was not evaluated for adequacy or maturity [69].

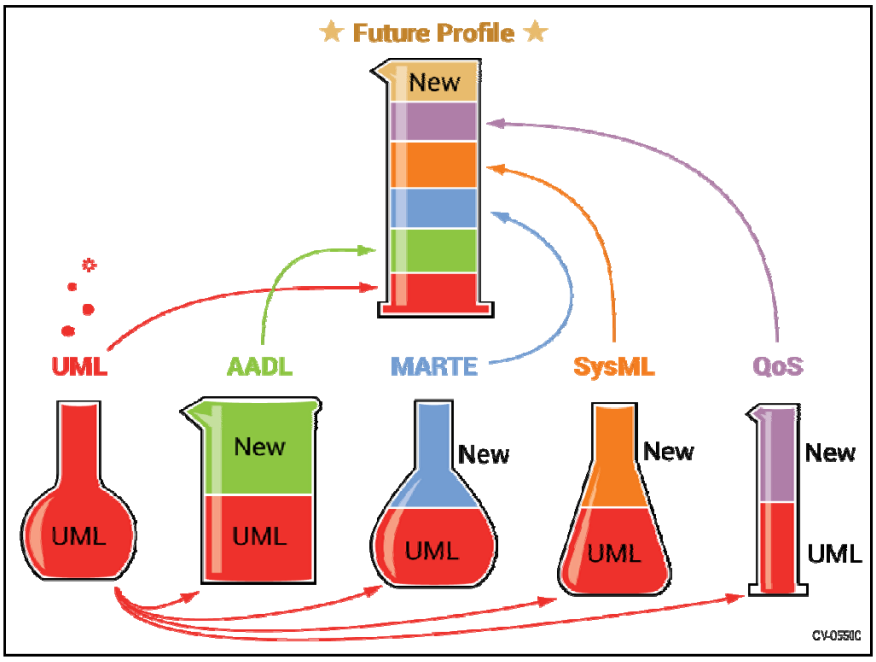


Figure 8. Safety-Critical UML Profile

V. CONCLUSION

This report provided a primer of UML, MBD/MDA®, and safety. Building on that brief information, a short list of UML's undesirable traits was provided and some recommended practices to mitigate those traits was suggested; for example, develop a UML design standard, select UML tool chain carefully, and conduct peer reviews, verification, and validation. Next, a sketch was provided for the development of a UML profile for safety-critical applications. Now, a request for feedback is being made to the safety-critical community using UML to determine the adequacy of these recommendations. In some cases, not enough constraints were suggested, while in other cases, the recommendations may be too difficult to address with current technology. In either situation, feedback and meetings are requested with those in the safety-critical industry who have used, are using, or planning to use UML for MBD and have gone, are going, or planning to go through certification or regulatory approval processes,

industry/commercial, civilian, or military. Going forward, the desire is to further refine these recommendations against lessons learned and continue to build a JSF- or MISRA-like standard for using UML in safety-critical applications. Putting in place those types of standards for UML will strengthen the case for the use of UML in MBD for safety-critical applications.

Remember that no single rule, practice, or standard alone is going to address making UML a fit for use on a safety-critical MBD application. Just like coding standards are one small part of an overall software safety program, UML design practices must be integrated into the whole software safety and system safety program. Moreover, to have a successful software safety program, rigorous creative disciplined effort must still be applied by the whole software safety team during the whole software life cycle. Do not become complacent once another best practice is produced. It is just one more tool to help have a successful software safety program.

REFERENCES

1. "Model Driven Architecture® (MDA®) FAQ," Object Management Group® (OMG®), http://www.omg.org/mda/faq_mda.htm.

2. Fowler, M. and Highsmith, J., "The Agile Manifesto," August 2001, http://www.pmp-projects.org/Agile-Manifesto.pdf.

3. Fowler, M. "The New Methodology," December 2005, http://www.martinfowler.com/articles/newMethodology.html.

4. Glazer, H. et al., "CMMI or Agile: Why Not Embrace Both!" Technical Note CMU/SEI-2008-TN-003, November 2008, http://www.sei.cmu.edu/library/abstracts/reports/08tn003.cfm.

5. "Rational Unified Process, Best Practices for Software Development Teams," Rational Software White Paper, TP026B, Revision November 2001, http://www.ibm.com/developerworks/rational/library/content/03July/1000/1251/1251_bestpractices_TP026B.pdf.

6. Dobolyi, K., "Software Requirements/Design Modeling," George Mason University (GMU), http://www.cs.gmu.edu/~kdobolyi/cs321/index.html.

7. Software Considerations in Airborne Systems and Equipment Certification, Radio Technical Commission of Aeronautics (RTCA) Document (DO) 178C, RTCA, Inc., Washington, DC, December 2012.

8. Model-Based Development and Verification Supplement to DO-178C and DO-278A, Radio Technical Commission of Aeronautics (RTCA) Document (DO) 331, RTCA, Inc., Washington, DC, December 2012.

9. "MDA® Specification," Object Management Group, Inc. (OMG®) http://www.omg.org/mda/specs.htm#MDAspecSupport.

10. "Object Management Group Terms and Acronyms," Object Management Group (OMG®), http://www.omg.org/gettingstarted/terms_and_acronyms.htm#M.

11. Mukerji J. and Miller, J., "MDA® Guide Version 1.0.1," June 2003, http://www.omg.org/cgi-bin/doc?omg/03-06-01.

12. "Documents Associated with Meta Object Facility™ (MOF™) Version 2.4.1," Object Management Group® (OMG®), June 2013, http://www.omg.org/spec/MOF/2.4.1/.

13. "Meta Object Facility™ (MOF™) Specification, Version 1.4.1," Object Management Group® (OMG®), July 2005, http://www.omg.org/spec/MOF/ISO/19502/PDF.

14. Guidelines from Development of Civil Aircraft and Systems, SAE ARP4754A, SAE Aerospace, Warrendale, PA, December 2010.

REFERENCES (CONTINUED)

15. Guidelines and Methods for Conducting the Safety Assessment Process on Civil Airborne Systems and Equipment,SAE ARP4761, SAE Aerospace, Warrendale, PA, December 1996.

16. "Model-Driven Development, IEEE Software," Institute of Electrical and Electronics Engineers (IEEE) Computer Society, 2003, http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=01231145.

17. "Object Management Group® Unified Modeling Language (OMG® UML)," International Organization for Standardization (ISO)/International Electrotechnical Commission (IEC) 19505-1:2012(E), Information Technology—Infrastructure, May 2012, http://www.iso.org/iso/catalogue_detail.htm?csnumber=32624.

18. "Object Management Group® Object Constraint Language (OMG® OCL)," International Organization for Standardization (ISO)/International Electrotechnical Commission (IEC) 19507:2012(E), Information Technology, May 2012, http://www.iso.org/iso/home/store/catalogue_tc/catalogue_detail.htm?csnumber=57306.

19. "Object Management Group® Unified Modeling Language (OMG® UML)," International Organization for Standardization (ISO)/International Electrotechnical Commission (IEC) 19505-2:2012 (E), Information Technology—Superstructure, May 2012, http://www.iso.org/iso/catalogue_detail.htm?csnumber=52854.

20. Unified Modeling Language: Diagram Interchange, Volume 2.0, September 2003.

21. "UML Diagram Overview.svg," September 2011, http://en.wikipedia.org/wiki/File:UML_diagrams_overview.svg.

22. Bell, D., "UML Basics – The Class Diagram," September 2004 http://www.ibm.com/developerworks/rational/library/content/RationalEdge/sep04/bell/.

23. Miles, R. and Hamilton, K., Learning UML 2.0, O'Reilly Media, Inc., May 2006.

24. Rierson, L., Developing Safety-Critical Software, CRC Press, Boca Raton, FL, 2013.

25. Schulmeyer, G. G., Handbook of Software Quality, Assurance, 4th Edition, Artech House, Inc., 2008.

26. "Department of Defense Standard Practice: System Safety," MIL-STD-882E, May 2012, http://www.system-safety.org/Documents/MIL-STD-882E.pd.

27. "NASA Technical Standard: Software Safety Standard," National Aeronautics and Space Administration (NASA) Standard (STD) 8719.13B, July 2004, http://www.hq.nasa.gov/office/codeq/doctree/871913.htm.

REFERENCES (CONTINUED)

28. "Software Requirements, Functional Safety of Electrical/Electronic/Programmable Electronic Safety-Related Systems," International Electrotechnical Commission (IEC) 61508-3, 2010.

29. "Overview of techniques and measures, Functional Safety of Electrical/Electronic/ Programmable Electronic Safety-Related Systems," International Electrotechnical Commission (IEC) 61508-7, 2010.

30. "Joint Strike Fighter Air Vehicle C++ Coding Standards for the System Development and Demonstration Program," Document Number 2RDU00001, Revision C, December 2005, http://www.stroustrup.com/JSF-AV-rules.pdf.

31. "MISRA-C: 2004 Guidelines for the Use of the C Language in Critical Systems," Motor Industry Software Reliability Association (MISRA), October 2008.

32. "MISRA-C++: 2008 Guidelines for the Use of the C++ Language in Critical Systems," Motor Industry Software Reliability Association (MISRA), July 2008.

33. "Jet Propulsion Laboratory (JPL) Institutional Coding Standard for the C Programming Language," Jet Propulsion Laboratory (JPL) DOCID D-60411, Volume 1.0, March 2009.

34. Anderson, D., Agile Management of Software Engineering: Applying the Theory of Constraints for Business Results, Prentice Hall, September 2003.

35. Adzic, G., "Improving Testing Practices at Google," December 2009. http://gojko.net/2009/12/07/improving-testing-practices-at-google/.

36. Boehm, B. W., "Software Engineering," International Conference on Software Engineering (ICSE) Proceedings of the 4th International Conference on Software Engineering, pp. 11-21, August 1976.

37. Gutz, S., "Static Analysis IBM Rational Software Analyzer: Getting Started," April 2008, https://www.ibm.com/developerworks/rational/library/08/0429_gutz1/.

38. Faugère, M. et al., "MARTE: Also an UML Profile for Modeling AADL Applications," Engineering Complex Computer Systems, 12th Institute of Electrical and Electronics Engineers (IEEE) International Conference, pp. 359-364, July 2007.

39. Faugère, M. et al., "Executing AADL Models with UML/MARTE," 14th Institute of Electrical and Electronics Engineers (IEEE) International Conference on Engineering of Complex Computer Systems, 2009.

40. Saadatmand, M. et al., "UML-Based Modeling of Non-Functional Requirements in Telecommunication Systems," International Conference on Software Engineering Advances (ICSEA) 2011: The Sixth International Conference on Software Engineering Advances, 2011.

REFERENCES (CONTINUED)

41. Delligatti, L., SysML Distilled: A Brief Guide to the Systems Modeling Language, First Edition, Addison-Wesley Professional, December 2013.

42. "Editing C++ code in Capsule-Based Models," http://pic.dhe.ibm.com/infocenter/rsarthlp/v7r5m1/index.jsp?topic=%2Fcom.ibm.xtools.rsdrt.modeling.code.doc%2Ftopics%2Ftovervieweditingcppcode.html.

43. Steimann F. and Vollmer, H., "Exploiting Practical Limitations of UML Diagrams for Model Validation and Execution," Springer Software and Systems Modeling, Volume 5, Issue 1, pp. 26-47, April 2006.

44. Tom Hill, Email concerning the differences between Rhapsody and Rational Software Architecture, 07 August 2013.

45. Tom Fox, Correspondence regarding document attachment title, "In Response to AI012 Feedback," from file GDAIS Responses to Advanced Information (AI) 012 Feedback.docx, 15 August 2012.

46. D. Bell, "UML Basics: An Introduction to the Unified Modeling Language," IBM, June 2003, http://www.ibm.com/developerworks/rational/library/769.html.

47. Warmer, J. and Klepp, A., The Object Constraint Language: Getting Your Models Ready for MDA®, Addison-Wesley, September 2003.

48. de Niz, D., "Diagrams and Languages for Model-Based Software Engineering of Embedded Systems: UML and AADL," Software Engineering Institute, Carnegie Mellon University, December 2007, http://www.sei.cmu.edu/library/assets/UML_AADL_Comparison.pdf.

49. Grady Booch, "Software Architecture, Software Engineering, and Renaissance Jazz, Microsoft and Domain Specific Languages," December 2004, https://www.ibm.com/developerworks/community/blogs/gradybooch/entry/microsoft_and_domain_specific_languages?lang=en.

50. Shousha, M. et al., "A UML/MARTE Model Analysis Method for Detection of Data Races in Concurrent Systems," Proceeding Models 2009, Proceedings of the 12th International Conference on Model Driven Engineering Languages and Systems, pp. 47-61, 2009, http://simula.no/research/se/publications/Simula.SE.645/simula_pdf_file.

51. Bradner, S., "Key Words for Use in RFCs to Indicate Requirement Levels," Harvard University, March 1997, http://www.ietf.org/rfc/rfc2119.txt.

52. Šilingas, D., "Best Practices for Applying UML, Part I," No Magic Inc., http://www.magicdraw.com/files/whitepapers/Best_Practices_for_Applying_UML_Part1.pdf.

REFERENCES (CONTINUED)

53. Object-Oriented Technology and Related Techniques Supplement to DO-178C and DO-278A, Radio Technical Commission for Aeronautics (RTCA) Document (DO) 332, RTCA, Inc., Washington, DC, December 1992.

54. Warmer J. and Kleppe, A., The Object Constraint Language: Getting Your Models Ready for MDA®, Second Edition, Addison-Wesley Professional, September 2003.

55. "AMCOM Software System Safety Policy," AMCOM 385-17, AMCOM Software System Safety Policy, United States (U.S.) Army Aviation and Missile Command, Redstone Arsenal, AL, March 2008.

56. Letelier, P., "A Framework for Requirements Traceability in UML-based Projects," Proceedings of First International Workshop on Traceability in Emerging Forms of Software Engineering, pp. 32-41, 2002.

57. Software Considerations in Airborne Systems and Equipment Certification, Radio Technical Commission of Aeronautics (RTCA) Document (DO) 178B, RTCA, Inc., Washington, DC, December 1992.

58. "IEEE Recommended Practice for Software Requirements" Institute of Electrical and Electronics Engineers (IEEE) Standard (STD) 830-1998, Specifications, International Standard Book Number (ISBN) 0-7381-0332-2, New York, NY, October 1998.

59. Balmelli, L., "An Overview of the Systems Modeling Language for Products and Systems Development," Journal of Object Technology, Volume 6, Number 6, July 2007, http://www.jot.fm/issues/issue_2007_07/article5/.

60. "Dia," https://wiki.gnome.org/Apps/Dia.

61. Merilinna, J. and Matinlassi, M., "Evaluation of UML Tools for Model-Driven Architecture," 11th Nordic Workshop on Programming and Software Development Tools and Techniques Nordic Workshop on Programming Environment Research (NWPER) 2004, page 155-162, Turku, Finland, August 2004, http://www.neone.fi/research/Evaluation_of_UML_Tools_for_Model-Driven_Architecture.pdf.

62. Soeken, M. et al., "Verifying Dynamic Aspects of UML Models," Design, Automation and Test in Europe Conference And Exhibition, March 2011, http://www.date-conference.com/proceedings/PAPERS/2011/DATE11/PDFFILES/09.3_1.PDF.

63. Ali, S. and Sulyman, M., "Applying Model Checking for Verifying the Functional Requirements of a Scania's Vehicle Control System," School of Innovation, Design, and Engineering Mälardalen University, Västerås, Sweden, September 2012, http://www.diva-portal.org/smash/get/diva2:558435/FULLTEXT01.

REFERENCES (CONCLUDED)

64. Daw, Z. et al., "Formal Verification of Software-Based Medical Devices Considering Medical Guidelines," International Journal of Computer Assisted Radiology and Surgery, Engineering in Medicine and Biology (EMB)-Laboratory, University of Applied Sciences—Mannheim, Baden-Württemberg, Germany, July 2013.

65. "Road Vehicles—Functional Safety—Part 6: Product Development at the Software Level," International Organization for Standardization (ISO) 26262-6:2011, November 2011, http://www.iso.org/iso/catalogue_detail?csnumber=51362.

66. "NASA Software Safety Guidebook," National Aeronautics and Space Administration (NASA) Guide Book (GB) 8719.13, NASA Technical Standard, March 2004, http://www.hq.nasa.gov/office/codeq/doctree/871913.pdf.

67. Noyrit, F. et al., "Consistent Modeling Using Multiple UML Profiles," Model Driven Engineering Languages and Systems, 13th International Conference, Models 2010, Proceedings, Part 1, Oslo, Norway, pp. 392-406, October 2010.

68. Espinoza, H. et al., "Challenges in Combining SysML and MARTE for Model-Based Design of Embedded Systems," Model Driven Architecture—Foundations and Applications, 5th European Conference, ECMDA-FA 2009 Enschede, Netherlands, pp. 98-113, June 2009.

69. Zoughbi, G. et al., "A UML Profile for Developing Airworthiness-Compliant (RTCA DO-178B) Safety-Critical Software," Model-Driven Engineering Languages and Systems, Lecture Notes in Computer Science, Volume 4,735, pp. 574-588, 2007.

LIST OF ABBREVIATIONS, ACRONYMS, AND SYMBOLS

0..*	zero or more
AADL	Architecture Analysis and Design Language
ASIL	Automotive Safety Integrity Level
CSID	Control Station Identification
CSType	Control Station Type
CWM	Common Warehouse Metamodel
DO	Document
DOORS®	Dynamic Object-Oriented Requirements System®
DSML	Domain Specific Modeling Language
GB	Guide Book
GCS	Ground Control Station
GPML	General Purpose Modeling Language
HW	Hardware
ID	Identification
IEC	International Electrotechnical Commission
INCOSE	International Council on Systems Engineering
int	Integer
ISO	International Organization for Standardization
JPL	Jet Propulsion Laboratory
JSF	Joint Strike Fighter
MARTE	Modeling and Analysis of Real Time and Embedded
MATLAB®	Matrix Laboratory®
MBD	Model-Based Development
MBSE	Model-Based Systems Engineering
MDA®	Model Driven Architecture®
MISRA	Motor Industry Software Reliability Association
MOF™	Meta-Object Facility™
NASA	National Aeronautics and Space Administration
NFP	Non-Functional Parameter
NFR	Non-Functional Requirement
OCL	Object Constraint Language

LIST OF ABBREVIATIONS, ACRONYMS, AND SYMBOLS (CONCLUDED)

OMG ®	Object Management Group®
OO	Object-Oriented
PIM	Platform-Independent Model
PSAC	Plan for Software Aspects of Certification
PSM	Platform-Specific Model
QoS	Quality of Service
RTCA	Radio Technical Commission of Aeronautics
RTVM	Requirement Traceability and Verification Matrix
RUP	Rational Unified Process
SDP	Software Development Plan
SRS	Software Requirements Specification
STD	Standard
SW	Software
SysML	System Modeling Language
UA, UAV	Unmanned Air Vehicle
UAID	Unmanned Air Vehicle Identification
UML	Unified Modeling Language

APPENDIX A
USING UNIFIED MODELING LANGUAGE IN MODEL-BASED DEVELOPMENT FOR SAFETY-CRITICAL APPLICATIONS

Using Unified Modeling Language in Model Based Development for Safety-Critical Applications

Sue Bonne
UAS SW Airworthiness Lead
AV Division, SED AMRDEC, RDECOM
susan.bonne@us.army.mil

Jason Rupert
SW Airworthiness Engineer
APT Research
jason.k.rupert@us.army.mil

Goal

Discuss topics and considerations for using the Unified Modeling Language (UML) in Model Based Development (MBD) for Safety-Critical Applications

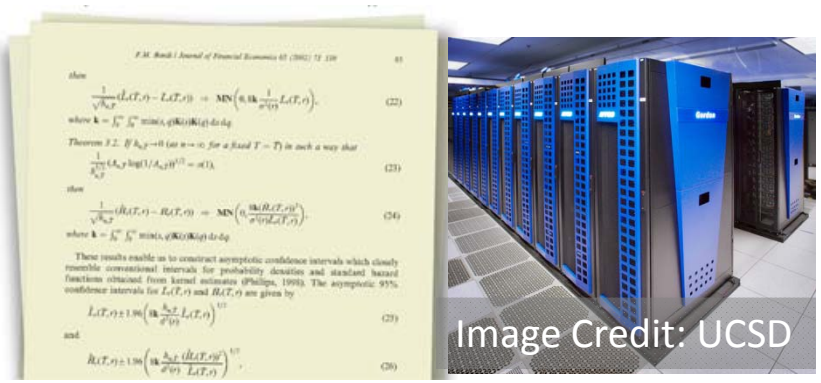
BLUF: No "silver bullet" – requires creative disciplined effort

Agenda

- Background
  - What is a model?
  - What is model based development?
  - What is UML?
  - What is safety and Safety-Critical SW*?
    - *Also applies to airworthiness
- Application
  - Models in MBD
  - UML in MBD
  - Safety to SW
- UML in Safety-Critical Applications
  - Some UML Concerns
  - Suggested Practices
    - UML Design Standards
    - UML MBD tool(s) chain selection
    - Safety/Airworthiness UML Profile
  - Reviews, Verification and Validation of UML Models
- Conclusions & Wrap-up
- Backup material
- References

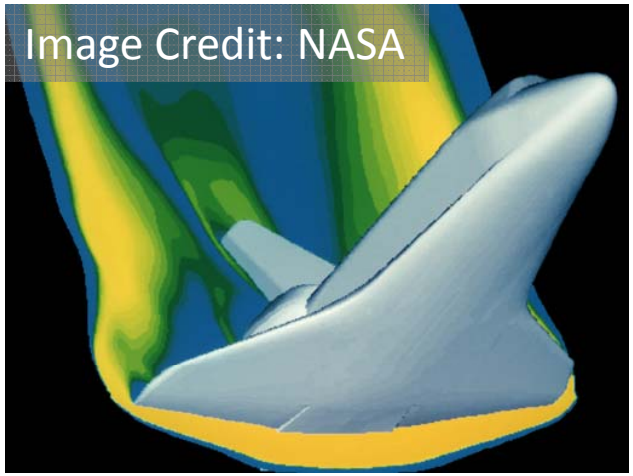
Caveat: This briefing only focuses on the use of UML in MBD. Should your MBD effort use a different general purpose modeling language (GPML) or domain specific language (DSL), similar recommendations would apply to its use, especially for safety-critical or flight critical efforts.

What are models?



Supercomputer Model of Stock Market High Frequency Trading

Computational



Physical

Team Run Production Model

RPSit = β1 + β2OBPit + β3SLGit + β4NL + eit

RPSit = number of runs produced by team i in season t.

OBPit = on-base percentage of team i in season t.

SLGit = slugging percentage of team i in season t.

NLi = dummy variable = 1 if team i is in the National League, 0 otherwise.

eit = random error for team i in season t.

$$h_s = \int_{T_1}^{T_2} c_P dt = \overline{c}_P (T_2 - T_1)$$

Mathematical model ~1895

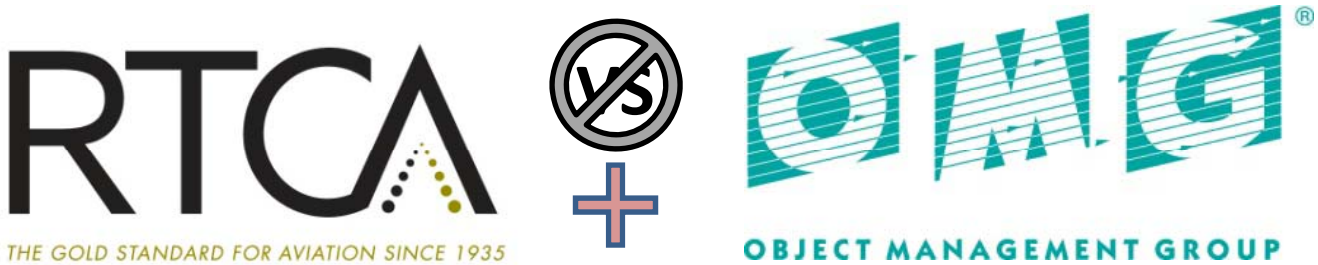
Enthalpy or total heat of superheated steam [1]

R. T. Kent, *Kent's Mechanical Engineers' Handbook: Power, Eleventh Edition, John Wiley & Sons, New York NY, 1937.*

- Range from Baseball Team Performance Models (a.k.a., Money Ball) to CFD (Computational Fluid Dynamics) using domain specific modeling languages (DSML) to general purpose modeling languages (GPML)

Model

- RTCA DO-331 Model Based Development and Verification Supplement to DO-178C and DO-278A
  - "An abstract representation of a given set of aspects of a system that is used for analysis, verification, simulation, code generation or any combination thereof. A model should be unambiguous, regardless of its level of abstraction."
    - Note 1: If the representation is a diagram that is ambiguous in its interpretation, this is not considered a model
    - Note 2: The "given set of aspects of a system" may contain all aspects of the system or only a subset."
- Object Management Group (OMG®)
  - "Models in the context of the [Model Driven Architecture] MDA Foundation Model are instances of [Meta-Object Facility] MOF™ metamodels and therefore consist of model elements and links between them." This required MOF compliance enables the automated transformations on which MDA is built. UML compliance, although common, is not a requirement for MDA models. (This means, for example, that a suitable development process based on OMG's Common Warehouse Metamodel can be MDA-compliant, since CWM is based on MOF.)" (http://www.omg.org/mda/specs.htm#MDAspecSupport)



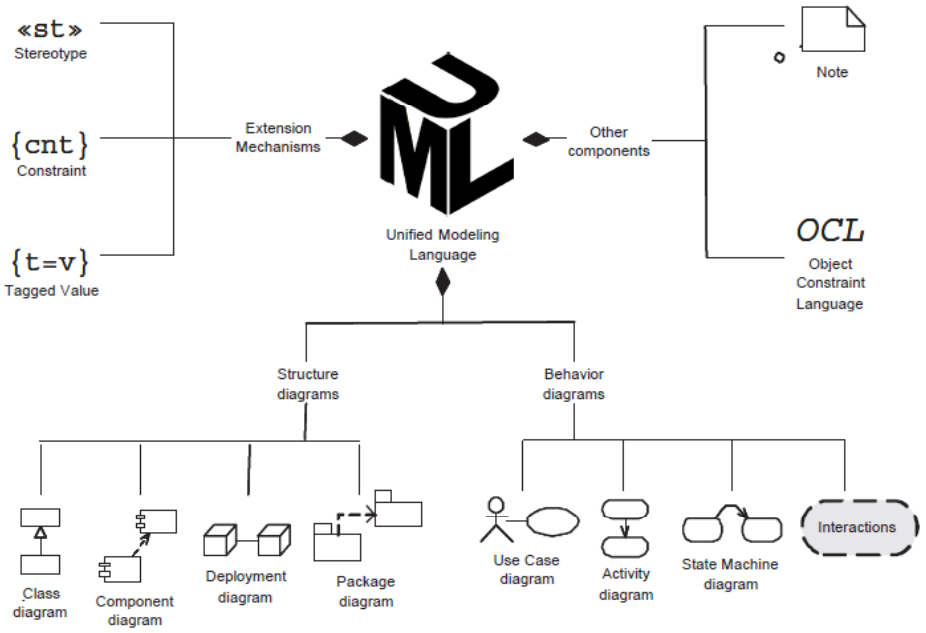
Model Based Development (MBD)

- RTCA, DO-331 Model Based Development and Verification Supplement to DO-178C and DO-278A
  - "A technology in which models represent software requirements and/or software design descriptions to support the development and verification process."
- Model Driven Architecture (MDA) is related to MBD
  - OMG® (http://www.omg.org/mda/faq_mda.htm#what%20is%20mda)
    - "The MDA is a new way of developing applications and writing specifications, based on a platform-independent model (PIM) of the application or specification's business functionality and behavior. A complete MDA specification consists of a definitive platform-independent base model, plus one or more platform-specific models (PSM) and sets of interface definitions, each describing how the base model is implemented on a different middleware platform. A complete MDA application consists of a definitive PIM, plus one or more PSMs and complete implementations, one on each platform that the application developer decides to support. "
    - Any modeling language used in MDA must be described in terms of the MOF language, to enable the metadata to be understood in a standard manner, which is a precondition for any ability to perform automated transformations.

"Model-driven development is *simply* the notion that we can construct a model of a system that we can then transform into the real thing." Model-Driven Development, IEEE Software, Published by the IEEE Computer Society, 0740-7459, 2003.

What is UML?

- Unified Modeling Language™ (UML) is a general purpose modeling language (GPML) that uses a set of graphic notations to specify, constrain and document visual models
- UML is managed by the OMG and is an ISO standard
- One key UML 2.x Specifications (of four)
  - Object Constraint Language (OCL)
    - Formal language lacks some of the ambiguities associated with natural languages
    - Defines invariant rules for model element, so, for example could be used to clearly describe required safety guard conditions and exit threshold inhibits
    - The “Big Six Diagrams" of UML fall into two categories
      - Static/structural: Use Case, Class and Composite Structure
      - Dynamic/behavioral: Activity, Sequence and State Machine (or State Diagrams)

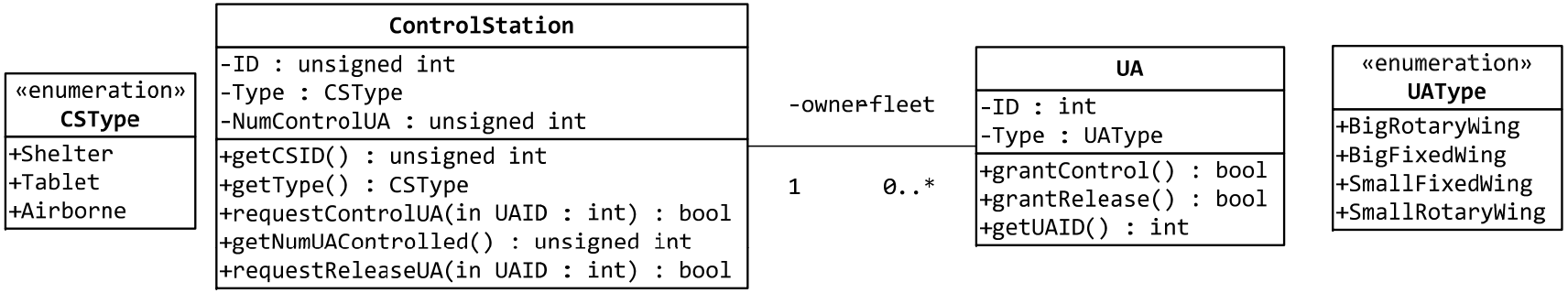


“UML 2.0 Interactions with OCL/RT Constraints”, Daniel Calegari Garcia, 2007.
http://www.fing.edu.uy/inco/pedeciba/bibliote/tesis/tesis-calegari.pdf

Two Separate Examples of UML

- **Example 1: UA Control Station Ownership**

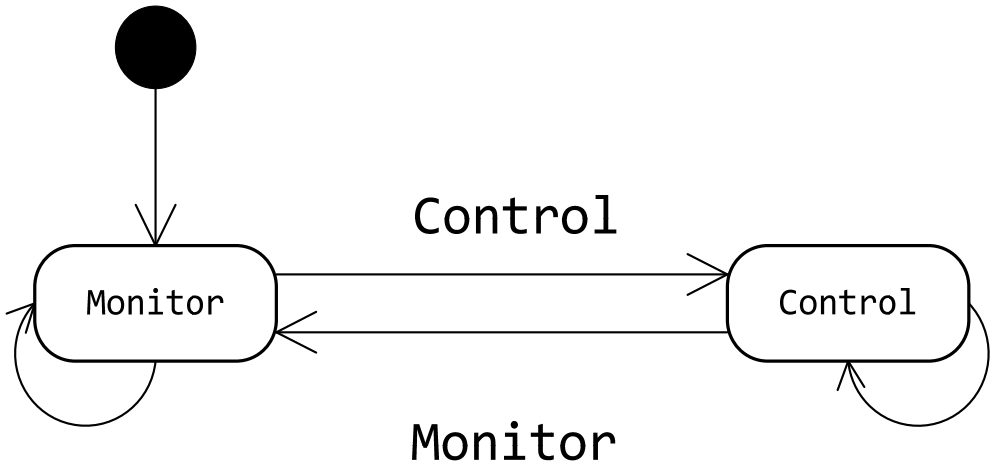
Class



UA = unmanned aircraft "*" = multiplicity; i.e., 0 to n, where n is a bunch

- **Example 2: Control Station State Transition**

State Machine



What is Safety and Safety-Critical SW?

- Safety
  - "freedom from those conditions [hazard] that can cause death, injury, illness, damage to or loss of equipment or property, or environmental harm", Handbook of Software Quality Assurance, Chapter 9 Software Safety and Its Relation to Software Quality Assurance, (Schulmeyer)
- Hazard (MIL-STD-882E)
  - A real or potential condition that could lead to an unplanned event or series of events (i.e. mishap) resulting in death, injury, occupational illness, damage to or loss of equipment or property, or damage to the environment.
- Safety-Critical SW (MIL-STD-882E)
  - Software that has been determined through analysis to potentially contribute to a hazard with catastrophic or critical mishap potential
    - Provides controls or mitigations for a hazard
    - Controls safety-critical functions
    - Processes safety-critical functions
    - Detects and reports, or takes corrective action, if the system reaches a specific hazardous state
    - Mitigates damage if a hazard occurs
    - Resides on the same system (processor) as safety-critical software AND is NOT partitioned from non-essential software

Airworthiness and SW Airworthiness is not explicitly addressed here, but the principles presented here are also applicable to flight critical environments.

Applications

UML in MBD

- "Although not formally required [for MBD], UML* is still a key enabling technology for the Model Driven Architecture and the basis for 99% of MDA development projects.", OMG MDA FAQ
- Code Generation**, "round tripping", and Verification & Validation

*What about SAE Architecture Analysis and Design Language (AADL)? Primarily, AADL is a unique standalone DSL. Efforts have been made to extend MARTE Profile to map to AADL, but adequate discussion of that effort is beyond the scope of this presentation.

** Code Generation Clarification: Code generation is a bit of a misnomer since the developer still actually writes the behavior code in the model. This code provides the "algorithmic details" that actually make the application work. Thus, appropriate low level requirements and design details are necessary for this hand-written behavior code that is included in the model.

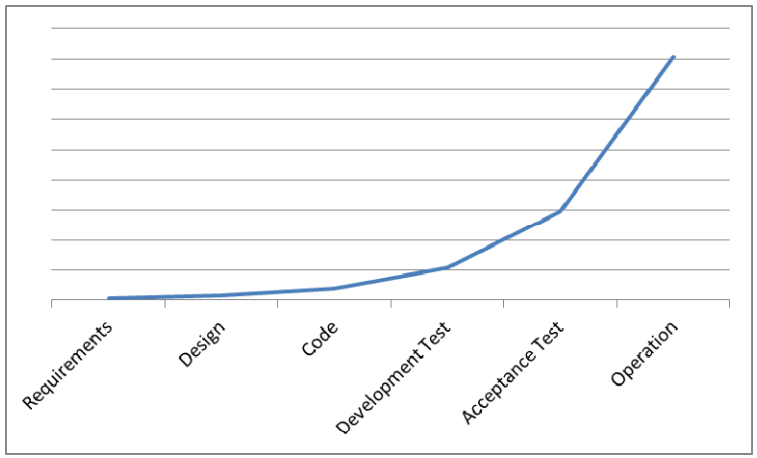
Models in MBD

- System Models
  - SysML Profile; i.e., UML Profile* developed by subsetting and extending UML
    - Requirements are included as “first-class model elements”, UML-Based Modeling of Non-Functional Requirements in Telecommunication Systems, M. Saadatmand, et al., 2011
    - Requirements table; i.e., “traceability information for requirements in a single view” (See above reference)
- PIMs and PSMs
- Low Level Models
  - Due to some of the inadequacies of UML, some choose to integrate Domain Specific Models (DSMs) with UML models
    - MatLab®/Simulink®
    - SCADE

*UML Profiles are analogous to dialects; e.g., Cajun is a dialect of North American English. UML Profiles can be composed of a namespace and a list of stereotype names, modeling conventions, constraints and tagged values.

Applications of Safety to SW

- Example Guiding Documents
  - Objectives, Activities, Artifacts (e.g., Plans and Standards) and Processes
    - IEEE 12207/IEC 61508
    - 882E/385-17
    - ARP4754/DO-178C
    - NASA-STD-8719.13B/C
    - ISO 26262
- Coding Standards
  - JPL Institutional Coding Standard for the C Programming Language
  - MISRA for C and C++
  - JSF for C++
  - C/C++ Coding Standard Recommendations for IEC 61508
  - Others (e.g., CERT)



Boehm Curve: relative cost to fix error verse phase in which error detected

- Weakness of Coding Standards
  - Come late in the SW Lifecycle (especially for MBD)
    - The later in the SW Lifecycle issues are caught and identified the more costly to fix (see Boehm Curve)
  - Don't address requirements and design verification and traceability issues (although they should, as required by DO-178C and also DO-178C requires Requirement Standards as well)
  - Most of the time only address static analysis concerns, not logical errors

Apply Safety early in the SW lifecycle

UML in MBD For Safety-Critical Applications

Some UML Concerns

- Informal
  - "UML is not a methodology, it does not require any formal work products", UML basics: An introduction to the Unified Modeling Language
  - "A UML diagram, such as a class diagram, is typically not refined enough to provide all the relevant aspects of a specification." OMG Object Constraint Language (OCL) Specification, Version 2.2, Section 7.2, http://www.omg.org/spec/OCL/2.2/
  - "All models, both PSM and PIM, should be consistent and precise, and contain as much information as possible about the system. This is where OCL can be helpful, because UML diagrams alone do not typically provide enough information." The object constraint language: getting your models ready for MDA. J. Warmer & A. Klepp, Addison-Wesley 2003.
- Consistency
  - "UML cannot fully define the relationships between diagrams. The diagrams are developed as separate entities that express different aspects of the software, not as parts of a common construct." Diagrams and Languages for Model-Based Software Engineering of Embedded Systems: UML and AADL, Dionisio de Niz, CMU SEI, 2007.
  - "the current UML spec really does not restrict graphical formats in any way -- it simply provides a standard set of notations, but not at the exclusion of other notations. In other words, there really is no "illegal" UML graphical syntax." Grady Booch [2004]
- Questionable Behavior
  - "In UML, active objects have their own thread of control, and can be regarded as concurrent threads . Only extensions of the UML standard, such as the MARTE profile, provide mechanisms to model detailed information pertaining to concurrency." A UML/MARTE Model Analysis Method for Detection of Data Races in Concurrent Systems

Some Considerations for UML in MBD For Safety-Critical Applications

Some Suggested Practices for using UML in a Safety-Critical MBD Application

- UML Design Standards (in addition to Requirements and Coding Standards)
  - Address good practices
  - Address model constraints and expressions
  - Address traceability, especially for safety-critical requirements
  - Address extensions, stereotyping, and non-functional parameters (NFPs/tagging)
- UML MBD tool(s) chain selection
  - Diagram consistency verification; i.e., verifies consistency among diagrams
  - Static/structural and Dynamic/behavioral verification
  - Constraint verification
  - Tool Qualification & Output Certification
- (Future) Safety/Airworthiness UML Profile*
  - Stereotypes
  - Constraints

*Isn't there already MARTE? Yes. But MARTE does not consider Safety-Critical concerns; e.g., traceability, or full use of constraints.

UML Design Standards Capture Good Practices

- To-dos and not to-dos; i.e., which UML constructs should always be done or avoided
- Examples
  - Graphical design indicators
    - Place a ***thick red border*** on safety-critical design elements (NASA-STD-8719.13B)
  - For class diagrams
    - Multiplicity (must be explicitly limited)
    - Anonymous instances (allowed or not)
    - Displaying attributes and operations/functions (to show or not to show)
    - Inheritance, polymorphism, abstraction, etc. (See DO-332 for other Object Oriented Programming concepts that must be addressed)
  - For state diagrams/charts (link provided by Ed Mayer, IBM)
    - No concurrent states; i.e., more than one state can be active at once; e.g., hierarchical state machine (HSM)
    - Do not use multiple entries and exits
    - Composite/nested substates (specify how many levels of nesting)
    - Compound transitions
- DO-331 Sec.11.23 provides a list of good practices that should be included in a Software Model Standard; e.g., maximum number of models per diagram, etc.

For more on state diagrams also see: http://www.statesoft.org/statemachinegenerator.html and http://sce.uhcl.edu/helm/rationalunifiedprocess/process/modguide/md_stadm.htm

UML Design Standards Capture Constrains

- Constraints – "a restriction on one or more values of (part of) an object-oriented model or system" The object constraint language: getting your models ready for MDA. 2003.
  - Informal; e.g., natural language
  - Formal; e.g., Object Constraint Language (OCL)
    - A declarative language, which is part of the UML Specification, for describing rules that apply to UML models. (from "The Object Constrain Language")
      - "you access an attribute, call a function, or select objects from a collection"
    - OCL is used to specify invariants of objects and preconditions and postconditions of operations.
    - Makes UML (class) diagrams more precise.
    - (Good) Unintended consequence: potentially can be used as a analysis or query language

OCL can be used to help specify the PIM to PSM transformation, "Implementing UML model transformations for MDA", Staron, M. and Kuzniarz, L., 2004

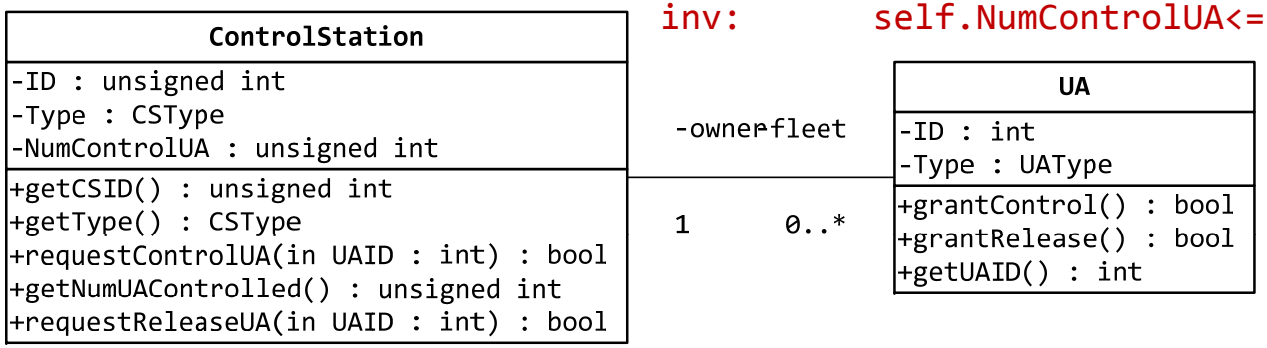
Two Separate Examples with Constraints

- **Example 1: UA Control Station Ownership**

Class

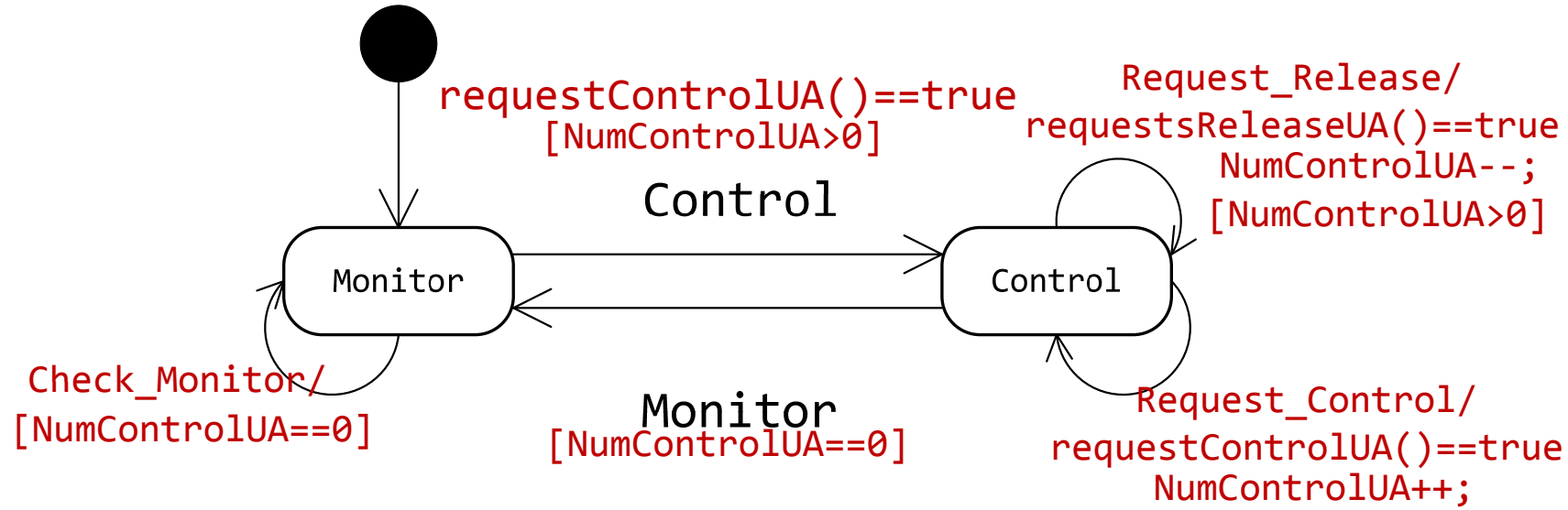
"A ControlStation owner shall control 3 or fewer UA.":

```
context ControlStation
inv:      self.NumControlUA<=3
```



- **Example 2: Control Station State Transition**

State Machine



UML Design Standards Capture Traceability

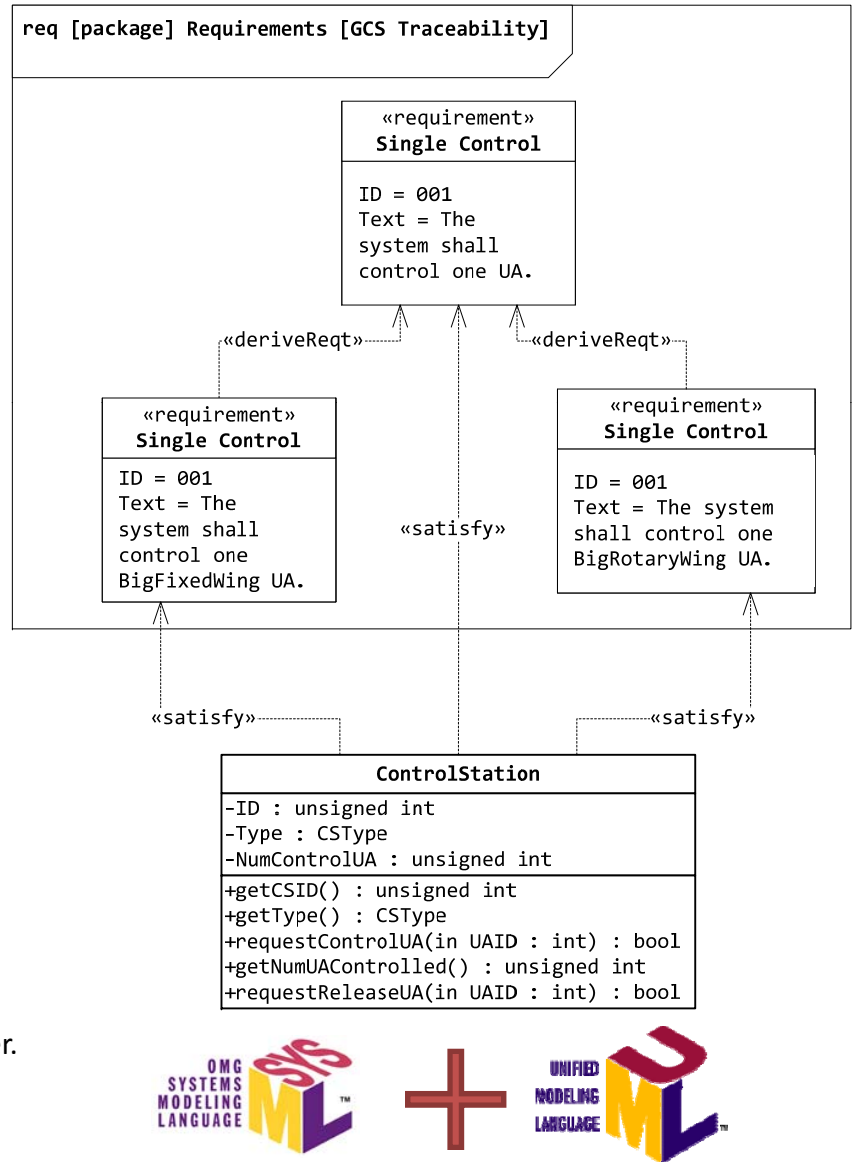
- What is Requirements Traceability?
  - "Requirements traceability is defined as the ability to describe and follow the life of a requirement in both directions, towards its origin or towards its implementation, passing through all the related specifications." A Framework for Requirements Traceability in UML-based Projects, Letelier, P.
  - RTCA DO-331 Model Based Development and Verification Supplement to DO-178C and DO-278A
    - MB.5.5 "When using model-based development, identification of requirements as per the method defined in the Software Model Standards [e.g., the UML Design Standards] should be used for bi-directional traceability. Means for this traceability should also be defined in the Software Model Standards."
    - MB.5.5"Since functional requirements are implemented using combinations of model elements, these combinations should therefore be used for bi-directional traceability."
    - MBA.11.23 states that the SW Model Standard should state the "method to be used to identify and delimit the derived requirements contained in the model and the method to provide derived requirements to the system process, including the system safety assessment process"

Note: IEEE 12207/IEC 61508 and 882E/385-17 call for a similar level of traceability.

UML Design Standards Capture Traceability

- Requirements traceability is not part of UML
- But (limited and still not formal) traceability can be added through the use of the OMG's SysML Profile
- Using SysML constructs enter the requirements and in a hierarchical fashion deconstruct them to the appropriate level
- Trace the those requirements to the appropriate UML Model element(s)
- SysML Requirements tables

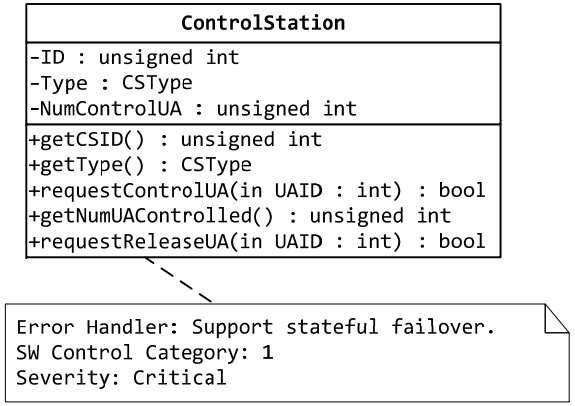
Note: Various modeling and requirements management tools have product specific (non-UML) ways of implementing this approach. However, some; e.g., Atego's Artisan Studio, use SysML to "allows users to take a DOORS requirements module and synchronize it into the model."



Example from "An Overview of the Systems Modeling Language for Products and Systems Development ", Laurent Balmelli, Ph.D., Manager, International Business Machine (IBM), Research Division, T.J.Watson Center. 2007.

UML Design Standards Capture Other Normative and Non-normative Notations

- Informal; e.g., natural language
  - For example, allowable use of the notes element
  - Capture rationale for design decisions
- Extensions (normative and non-normative); e.g., stereotypes
    - UML 2.1 contains 28 stereotypes
    - Unique profiles most likely will add additional profiles; e.g., <<OS>>, <<database system>>, << tool generated>> and <<custom code>>
      - For example, SysML adds 5 stereotypes: <<conform>>, <<view>>, <<viewpoint>>, <<rationale>> and <<problem>>
- Rules for tagged value attributes on an added stereotype
- Non-functional parameters (NFPs); e.g., safety, security, reliability, and performance
  - Address how are NFPs are to be described and attached to UML model elements



UML MBD tool(s) chain selection

- Assure UML and OCL interchange level
  - Supports highest model interchange compliance level, as defined in the OMG UML Superstructure specification (see UML v. 2.4.1, Chapter 2)
  - Same for OCL
- Example UML MBD Tool Evaluation Criteria
  - Announced UML version?
  - Support full UML specification; e.g., extensions and OCL?
  - Provide extensibility interface for user defined plug-ins?
  - Interface with Domain Specific Models or Domain Specific Languages?
  - Perform inter-diagram consistency verification; i.e., verifies consistency among diagrams?
  - Perform static/structural constraint verification?
  - Perform dynamic/behavioral constraint verification?
  - Is the tool qualifiable; e.g., according to DO-330 Tool Qualification?
  - Does the tool produce certifiable code?
- Some tool evaluation results are provided in the following
  - "Evaluation of UML Tools for Model-Driven Architecture", page 155-162, 11th Nordic Workshop on Programming and Software Development Tools and Techniques NWPER'2004 held in Turku, Finland, August 17-20, 2004.

Listing or endorsing specific tools or approaches, or evaluation of specific tool capabilities is beyond the scope of this briefing.

Review, Verification and Validation of the UML

Similar to the reviews, verification and validation of coding/programming; e.g., automated static analysis

Reviews, Verification and Validation of UML Suggested Practices

- Peer Reviews (informal and formal to catch errors before they are implemented and make use of good practices)
  - Design abides by the UML Design Standards; e.g., traceability
  - Look for things not easily found by automated reviews; e.g., good use of abstraction, modularity, cohesion, and coupling
- Analysis of static and dynamic view of UML model*
- Verifying Consistency
  - Consistency amongst static and dynamic views
    - For example, constraints in the UML class diagram are being followed in the state charts/diagrams
  - Consistency amongst the many views

*Please don't ask me to explain these. Reference papers to these techniques can be provided.

Reviews, verification and validation processes should be explained and documented within the SW Lifecycle Planning Materials; e.g., Software Development Plan, or Plan for Software Aspects of Certification (PSAC).

UML Suggested Practices: Model Checking

- Model checking is a verification method that automatically and exhaustively checks that a model meets a given specification
- NASA Software Safety Guidebook, NASA-GB-8719.13, 2004.
  - reachability (does as system ever reach a certain state)
  - lack-of-deadlock (is deadlock avoided in the system)
  - safety (nothing bad ever happens)
  - liveness (something good eventually happens)
- Within UML tools model checking is limited
  - Export from UML tools to other dedicated model checking tools
  - Hopefully UML tools will rise to the occasion as done with MATLAB/Simulink Design Verifier*
    - Include formal methods
  - Disappointing to see UML tools lag behind DSL tools

* Reference: "Applying Model Checking for Verifying The Functional Requirements of a Scania's Vehicle Control System", September, 2012

Safety/Airworthiness UML Profile

Extend or combine current profile(s) or create a new profile applicable to the Safety-Critical and Airworthiness Domain; e.g., pieces of SysML + pieces of MARTE + Safety & Airworthiness extensions & constraints.

Reference: “UML-Based Modeling of Non-Functional Requirements in Telecommunication Systems”, M. Saadatmand et al. ICSEA 2011 : The Sixth International Conference on Software Engineering Advances, 2011.

Characteristics of a UML Profile

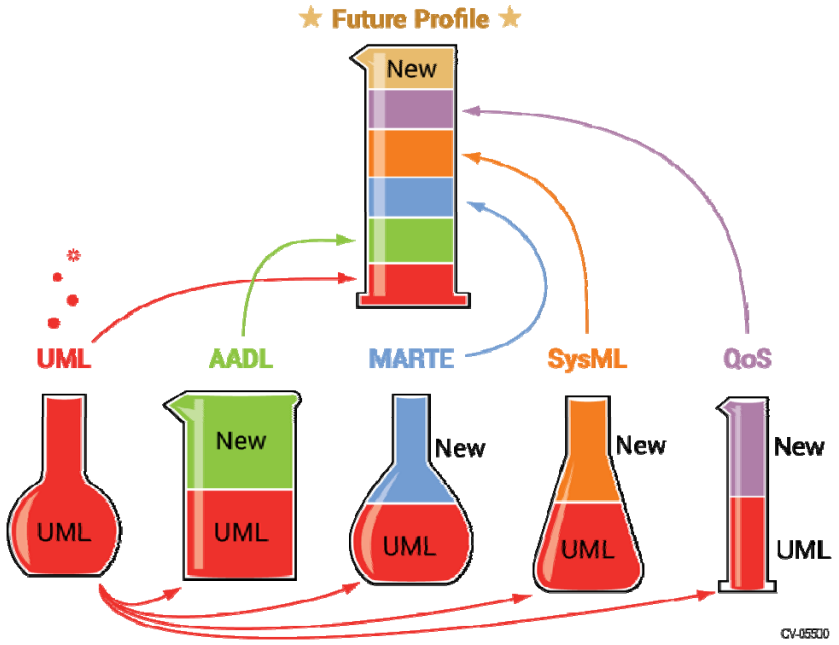
- Object Management Group Unified Modeling Language (OMG UML), Superstructure, ISO/IEC 19505-2:2012(E)
- Section 18.1.2 ISO/IEC 19505-2:2012(E) Profile Design Requirements
  - "Well-formedness"; e.g., "constraints that are more constraining (but consistent with) those in the reference [UML] metamodel" (that is, only extensions to the metamodel)
  - Using UML XMI it must be possible to interchange profiles between tools
  - Reference domain-specific UML libraries
  - Combine UML profiles and model libraries via extensions
  - Specialize semantics of standard UML elements; e.g., restrict to single inheritance without having to assign explicit stereotype to each and every instance
  - Profiles can be dynamically applied or retracted from a model

(Future) Safety/Airworthiness UML Profile

In UML extensions are an association relationship that indicates the properties of the metaclasss. Thus, quotes are used in the figure to indicate natural language definition is implied.

http://www.uml-diagrams.org/profile-diagrams.html#stereotype

NFP = non-functional parameters



- Figure shows a sketch of how this might look
- AADL Profile still seems to be under development
- Not enough is known about Security Profiles; e.g., UMLSec, to endorse one over another (Same comment applies to QoS Profile)
- Some known deficiencies with basic UML and MARTE have already been discussed
- Some effort is being undertake to develop DO-178 Airworthiness UML Profile, but it was not evaluated for adequacy or maturity. For example, reference: “A UML Profile For Developing Airworthiness-Compliant (RTCA DO-178B) Safety-Critical Software”, Gregory Zoughbi, Thesis, Carleton University, 2006.

Recall: No “silver bullet” – requires creative disciplined effort

Wrapping Up

"With great power there must also come great responsibility."
– Voltaire via Uncle Ben (Spiderman)

Conclusions

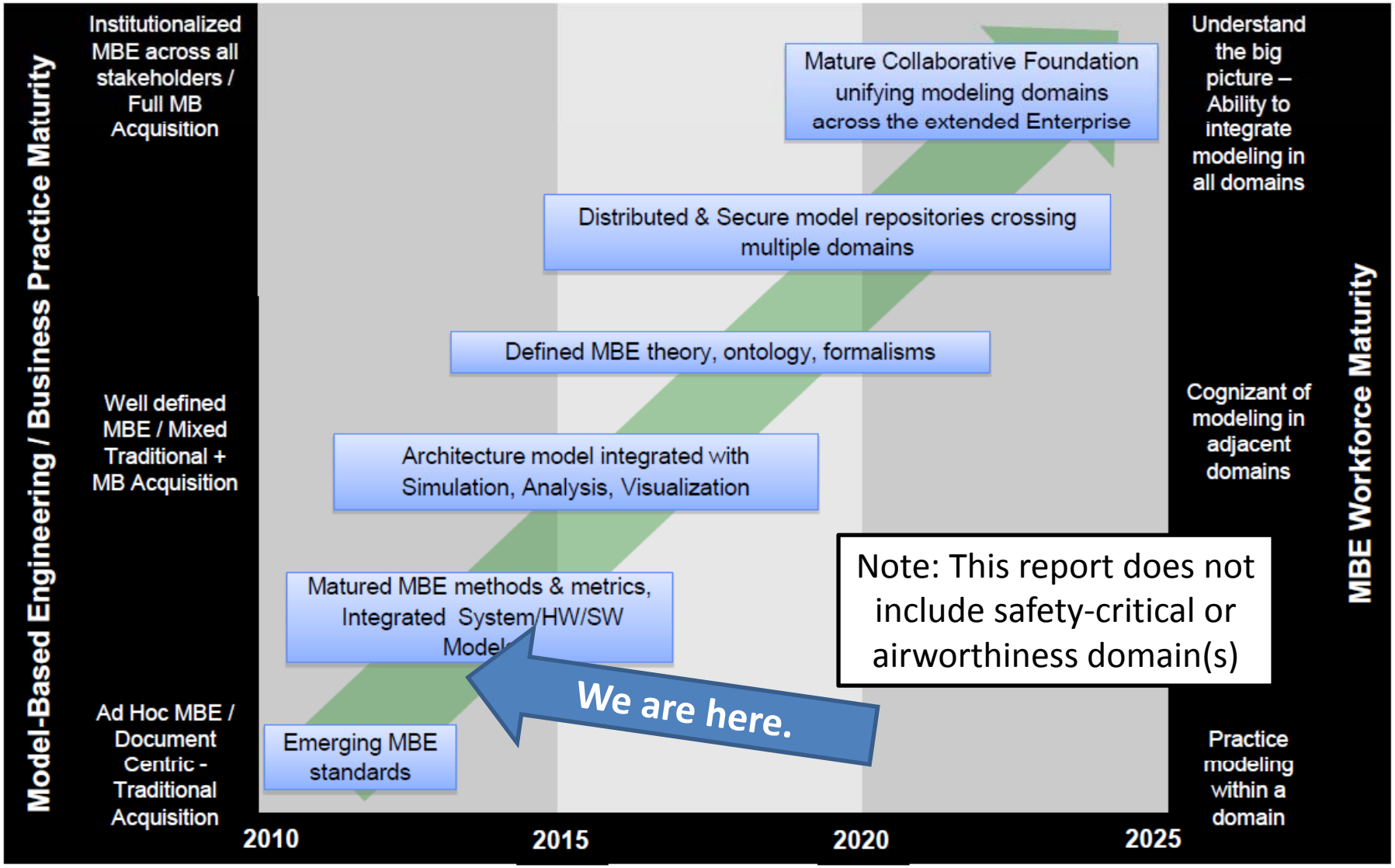
- Due to some of UML's characteristics as a powerful general purpose modeling language measures are necessary for it to be used in safety-critical MBD applications
  - Example measures:
    - Develop UML Design Standards
    - Choose adequate tool chain
    - Review, verify and validate UML Models
- Upon taking those measures, and properly planning for the use of MBD in the SW Lifecycle, the use of UML should be possible

Backup

Other information

MBE Roadmap

Reference: Final Report Model Based Engineering (MBE) Subcommittee, Jeff Bergenthal (Subcommittee Lead) , NDIA Systems Engineering Division, M&S Committee , February 2011



Source: INCOSE MBSE Roadmap

Requirements Traceability

- IEEE standard 830-1984 [19] states that:
  - A software requirements specification is traceable if (i) the origin of each of its requirements is clear and if (ii) it facilitates the referencing of each requirement in future development or enhancement documentation
- RTCA DO-178 B/C, Section 5.5 “bi-directional” trace data
  - “between the system allocated to software and high-level requirements”
  - “between the high-level requirements and low-level requirements (and architectural design decisions)”
    - Also, “when low-level requirements or software architecture are pressed by a Design Model, this model should conform to the Software Model Standards and be traceable, verifiable, and consistent.” DO-331 Sec. 5.2.2
    - “When high-level requirements are expressed by a Specification Model, this objective also aims at ensuring that no low-low level requirement traces to model elements that do not represent high-level requirements.” DO-331 Sec. 6.3.2 (f)
  - “between the low-level requirements and source code”
    - “When low-level requirements are expressed by a Design Model, this objective also aims at ensuring that no Source Code traces to model elements that do not represent low-level requirements.” DO-331 Sec. 6.3.4 (e)

Coverage Analysis

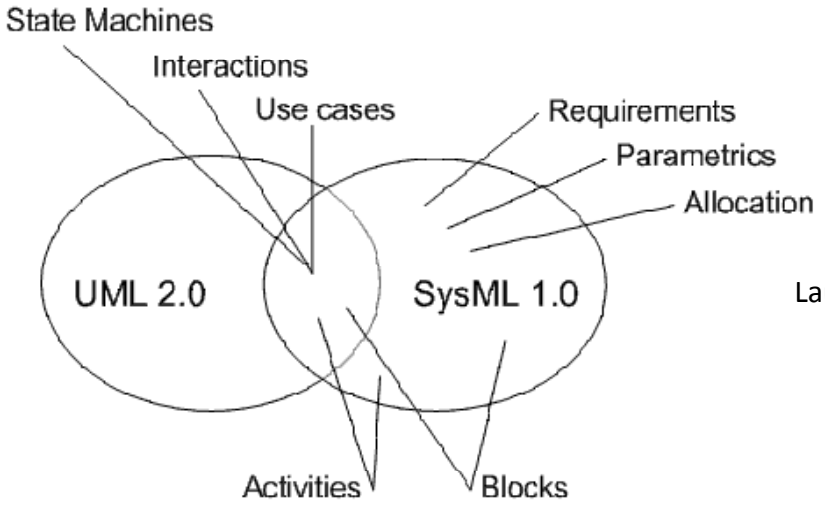
- RTCA DO-178 B/C, Section 6.4.4.1 For structural coverage: “test cases exist for each software requirement”
- Model Coverage Analysis DO-331 Section 6.7
  - If MBD with a Design Model is used, then both Model Coverage and Structural Coverage are required
  - Model Coverage should be performed using requirements-based verification using requirements from which the Design Model was developed

More UML Details

- "UML is a key enabling technology for Software Developers and Software Engineers who seek to transition from traditional, human-intensive, code-centric software development processes to Model-Driven Development (MDD) processes that are requirements-driven and architecture-centric." UML FAQ

SysML Profile

- Managed by OMG (http://www.omgsysml.org/)
- SysML Block Definition Diagram (BDD) is equivalent to the Class Diagram in UML



Laurent Balmelli: "An Overview of the Systems Modeling Language for Products and Systems Development", in *Journal of Object Technology, vol. 6, no. 6, July-August* 2007, pp. 149-177
http://www.jot.fm/issues/issue_2007_07/article5

Additional References

- "A UML Profile For Developing Airworthiness-Compliant (RTCA DO-178B) Safety-Critical Software", Gregory Zoughbi, Thesis, Carleton University, 2006.
- "Model-Driven User Requirements Specification using SysML", Journal of Software, VOL. 3, NO. 6, JUNE 2008.

USE: A Tool for Verifying and Validating UML

- USE (UML Specification Environment)
- available at: http://www.db.informatik.uni-bremen.de/projects/USE/
- A tool for validating OCL specifications
  - The developer can create test cases and check if the specified constraints are satisfied for these test cases
  - USE checks the test cases with respect to invariants and pre- post-conditions
- There are special USE commands for creating and manipulating object diagrams that can be accumulated in command files
- There is some support for automated testing
  - USE has a snapshot sequence language and a snapshot generator
  - The snapshot sequence language enables the user to write high level implementations for the user defined operations, so that they can be tested

See “CIS 771: Software Specifications”, Lecture 11: Introduction to OCL & USE

Analysis of static view of UML model*

- Consistency, independence, and consequences
  - M. Gogolla, M. Kuhlmann, and L. Hamann, “Consistency, Independence and Consequences in UML and OCL Models,” in TAP, ser. Lecture Notes in Computer Science, C. Dubois, Ed., vol. 5668. Springer, 2009, pp. 90–104.
- Enumerative methods
  - M. Gogolla, F. B¨uttner, and M. Richters, “USE: A UML-based specification environment for validating UML and OCL,” Science of Computer Programming, vol. 69, no. 1-3, pp. 27–34, 2007.
- Theorem provers
  - M. Kyas, H. Fecher, F. S. de Boer, J. Jacob, J. Hooman, M. van der Zwaag, T. Arons, and H. Kugler, “Formalizing UML Models and OCL Constraints in PVS,” Electronic Notes in Theoretical Computer Science, vol. 115, pp. 39–47, 2005.
- Constraint-Satisfaction-Problem (CSP) solvers
  - J. Cabot, R. Claris´o, and D. Riera, “Verification of UML/OCL Class Diagrams using Constraint Programming,” Apr. 2008, pp. 73–80.
- Boolean satisfiability (SAT)
  - M. Soeken, R. Wille, M. Kuhlmann, M. Gogolla, and R. Drechsler, “Verifying UML/OCL models using Boolean satisfiability,” in Design, Automation and Test in Europe. IEEE Computer Society, 2010, pp. 1341–1344.

*List is from “Verifying Dynamic Aspects of UML Models”, Soeken, M. ; Wille, R. ; Drechsler, R., Design, Automation & Test in Europe Conference & Exhibition 2011, Digital Object Identifier : 10.1109/DATE.2011.5763177

Analysis of dynamic view of UML model*

- Abstract State Machines (ASMs)
  - "A Toolset for Supporting UML Static and Dynamic Model Checking", W. Shen, K. Compton, J. Huggins, Proceeding COMPSAC '02 Proceedings of the 26th International Computer Software and Applications Conference on Prolonging Software Life: Development and Redevelopment, Pages 147-152, IEEE Computer Society Washington, DC, USA 2002.

*List is from "Verifying Dynamic Aspects of UML Models", Soeken, M. ; Wille, R. ; Drechsler, R., Design, Automation & Test in Europe Conference & Exhibition 2011, Digital Object Identifier : 10.1109/DATE.2011.5763177

“From UML diagrams to behavioural source code” (Thesis) 2007

- “Generation of behavioural source code can be done by means of UML behaviour diagrams. However, UML tools do not support the generation of source code from these diagrams, because (1) there is no one to one mapping between behaviour descriptions and the source code, (2) behaviour does not always have to appear in source code as explicit statements, (3) behaviour diagram can be implemented in different ways, and (4) the mapping between structure and behaviour diagrams is not always clear.”

- “A modeling language simply defines a grammar—a set of rules that determines whether a given model is well-formed or ill-formed. Those rules do not dictate how and when to use the language to create a model; they stop short of dictating any particular modeling method.” [SysML Distilled: A Brief Guide to the Systems Modeling Language]
- Dr. Frederick Brooks, Jr. in The Design of Design (Boston: Addison-Wesley, 2010), “Constraints are friends.” (p. 127).

INITIAL DISTRIBUTION LIST

		Copies
Defense Systems Information Analysis Center SURVICE Engineering Company 4695 Millennium Drive Belcamp, MD 21017	Ms. Jessica Owens jessica.owens@dsiac.org	Electronic
Defense Technical Information Center 8725 John J. Kingman Rd., Suite 0944 Fort Belvoir, VA 22060-6218	Mr. Jack L. Rike jackie.l.rike.civ@dtic.mil	Electronic
AMSAM-L	Ms. Anne C. Lanteigne hay.k.lanteigne.civ@mail.mil	Electronic
	Mr. Michael K. Gray michael.k.gray7.civ@mail.mil	Electronic
RDMR		Electronic
RDMR-CSI		Electronic
RDMR-AEV	Mr. David Hunnicutt david.a.hunnicutt2.civ@mail.mil	Electronic
	Mr. Josh Preusser joshua.j.preusser.civ@mail.mil	Electronic
	Mr. John Sims john.r.sims28.civ@mail.mil	Electronic
RDMR-BAV	Ms. Susan M. Bonne susan.m.bonne.civ@mail.mil	Electronic/Hardcopy
	Mr Phillip Howard phillip.howard.civ@mail.mil	Electronic
	Mr. Jonathan C. McNeil jonathan.c.mcneil.civ@mail.mil	Electronic
APT Research, Inc. 4950 Research Park Huntsville, AL 35805	Ms. Rhonda S. Barnes rhonda.s.barnes6.ctr@mail.mil	Electronic
	Ms. Melissa A. Emery melissa.a.emery6.ctr@mail.mil	Electronic
	Mr. Jason K. Rupert jason.k.rupert.ctr@mail.mil	Electronic
OTC 7501 South Memorial Parkway Suite 109 Huntsville, AL 35802	Mr. Enrique Ramos enrique.ramos2.ctr@mail.mil	Electronic

INITIAL DISTRIBUTION LIST (CONCLUDED)

		Copies
PPT Solutions 4825 University Square Suite 6 Huntsville, AL 35816	Ms. Amy E. Stratz amy.e.stratz.ctr@mail.mil	Electronic